UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

ANA BEATRIZ FERNANDES HERTHEL

Uma Abordagem Heurística para o *Single-finger Keyboard Layout Problem*

JOÃO PESSOA
2018

ANA BEATRIZ FERNANDES HERTHEL

Uma Abordagem Heurística para o *Single-finger Keyboard Layout Problem*

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal da Paraíba, como requisito parcial para a obtenção do grau de Mestre em Engenharia de Produção.

Orientador: Prof. Dr. Anand Subramanian

JOÃO PESSOA
2018

Catalogação na publicação
Seção de Catalogação e Classificação

H574a Herthel, Ana Beatriz Fernandes.
Uma abordagem heurística para o single-finger keyboard layout problem / Ana Beatriz Fernandes Herthel. - João Pessoa, 2018.
77 f. : il.

Orientação: Anand Subramanian.
Dissertação (Mestrado) - UFPB/CT.

1. Engenharia de produção. 2. Layout de teclado - Dedo único. 3. Teclados virtuais - Um dedo. 4. Atribuição quadrática - Problema. I. Subramanian, Anand. II. Título.

UFPB/BC

Ana Beatriz Fernandes Herthel

Uma Abordagem Heurística para o *Single-finger Keyboard Layout Problem*

Esta Dissertação foi julgada e aprovada em sua forma final para obtenção do grau de Mestre em Engenharia de Produção pelo Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal da Paraíba.

BANCA EXAMINADORA

Anand Subramanian

Prof. D.Sc. Anand Subramanian — Orientador
Departamento de Sistemas de Computação — UFPB

Prof. D.Sc. Lucídio dos Anjos Formiga Cabral — Examinador Interno
Departamento de Computação Científica — UFPB

Artur Alves Pessoa

Prof. D.Sc. Artur Alves Pessoa — Examinador Externo
Departamento de Engenharia de Produção — UFF

João Pessoa, 14 de março de 2018.

"The good thing about science is that it's true whether or not you believe in it."

Neil deGrasse Tyson

A minha mãe.

Agradecimentos

Este trabalho foi desenvolvido através de muita dedicação, esforço e renúncias, e não seria possível sem o apoio de muitas pessoas em suas mais diversas formas. Primeiramente agradeço a minha mãe Ana Margarethe por seu apoio incondicional e a meu irmão Nilton e sobrinho Lucas por suas presenças carinhosas. A meu marido Antônio Flávio pelo apoio, compreensão e socorros prestados durante esta difícil jornada. Agradeço a Anand por me apresentar a Pesquisa Operacional e o mundo da otimização, pela confiança, orientação e, acima de tudo, pela amizade tão querida. Agradeço aos professores Lucídio e Artur pelas sugestões à dissertação e disponibilidade em ajudar. Aos professores do PPGEP, por contribuírem com minha formação e a todos os funcionários do DEP pelo trabalho e colaboração. A meus colegas de laboratório (os que estão e os que já saíram): João, Carlos, Ewerton, Geraldo, Anderson, Crispim, Rubão, Vitor, Eduardo, Yuri, Fábio, Nailson, Luciano e Teobaldo, agradeço pelas ajudas, partidas de Uno e conversas de todos os dias que ajudaram a aliviar a ansiedade recorrente. A Juliana, Mayara, Luana Marques, Luana Leal, Giane, Marina, Débora, Brunna e a todos os colegas do mestrado que ajudaram a dividir as angústias de estar no mesmo barco. A minha família e amigos, agradeço sempre a compreensão e a torcida em todas as etapas da minha vida.

Resumo

A popularização de dispositivos móveis e outros aparelhos com teclados virtuais utilizados com um dedo revela o fato de que o *layout* QWERTY, desenvolvido originalmente para uso com os dez dedos das mãos, não atende às necessidades dos usuários. O problema associado ao desenvolvimento de um *layout* de teclado para um dedo é denominado SK-QAP e foi formalmente apresentado na literatura como uma variante do *Quadratic Assignment Problem* (QAP), um problema clássico de otimização conhecido por sua dificuldade de resolução. Uma revisão da literatura foi conduzida para reunir trabalhos relacionados ao desenvolvimento de teclados para um dedo e para n dedos que empregassem métodos vinculados à Pesquisa Operacional. Este trabalho utiliza uma abordagem heurística para resolver o SK-QAP através de um algoritmo *Iterated Local Search* (ILS), chamado ILS-SKQAP. Três estruturas de vizinhança foram incorporadas à fase de busca local do algoritmo. Duas delas (*contour filling* e *pairwise-exchange*) já utilizadas na resolução do SK-QAP, enquanto que a estrutura *two pairs swap* foi adaptada, neste trabalho, de uma vizinhança do QAP. Além disso, dois mecanismos de perturbação foram desenvolvidos para o ILS-SKQAP: *ejection chain* e *multiple pairwise-exchange*. O ILS-SKQAP foi usado na resolução das 24 instâncias existentes do SK-QAP para os idiomas inglês, francês, italiano e espanhol, obtendo resultados altamente competitivos em termos de qualidade das soluções encontradas e tempos computacionais. Ademais, seis novas instâncias foram desenvolvidas para a língua portuguesa e resolvidas pelo algoritmo.

Palavras-chave: *Single-finger*. *Layout*. Teclado. *Quadratic Assignment Problem*. *Iterated Local Search*.

Abstract

The popularization of mobile devices and other equipments with virtual single-finger keyboards unveils the fact that the QWERTY layout, originally developed for typing with all ten fingers, is not suited for the users' needs. The problem associated with the design of a single-finger keyboard layout is denoted SK-QAP and it was formally introduced in the literature as a variation of the Quadratic Assignment Problem (QAP), a classical and challenging optimization problem. A literature review was conducted to gather related work regarding single-finger and n finger keyboard layouts' design that employ an Operations Research-based methodology. This work proposes a heuristic approach to solve the SK-QAP by means of an Iterated Local Search (ILS) algorithm, called ILS-SKQAP. Three neighborhood structures were incorporated in the local search fase of the algorithm. Two of them (contour filling and pairwise-exchange) were previously applied to solve the SK-QAP, whereas two pairs swap was adapted, in this work, from a QAP neighborhood. Furthermore, two perturbation mechanisms were developed for the ILS-SKQAP: ejection chain and multiple pairwise-exchange. ILS-SKQAP was used to solve the 24 existing instances for English, French, Italian and Spanish languages with highly competitive results both in terms of solution quality and CPU time. Moreover, a set of six instances for the Portuguese language was developed and solved by the algorithm.

Keywords: Single-finger. Layout. Keyboard. Quadratic Assignment Problem. Iterated Local Search.

Sumário

Glossário

ACO	:	*Ant Colony Optimization*
ATOMIK	:	*Alphabetically Tuned and Optimized Mobile Interface Keyboard*
FANT	:	*Fast Ant System*
GA	:	*Genetic Algorithm*
ILS	:	*Iterated Local Search*
PSO	:	*Particle Swarm Optimization*
QAP	:	*Quadratic Assignment Problem*
RVND	:	*Random Variable Neighborhood Descent*
SA	:	*Simulated Annealing*
SK-QAP	:	*Single-finger keyboard layout problem*
SVNS	:	*Skewed Variable Neighborhood Search*
TS	:	*Tabu Search*
VND	:	*Variable Neighborhood Descent*
VNDS	:	*Variable Neighborhood Decomposition Search*
VNS	:	*Variable Neighborhood Search*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Definição do tema

Um teclado é um equipamento destinado a promover a entrada de dados em um dispositivo eletrônico. É formado por um conjunto teclas que produzem diferentes caracteres ao serem ativadas. Um *layout* de teclado configura o arranjo de símbolos numa malha de teclas, de forma que diferentes arranjos de caracteres podem ser obtidos a partir de uma mesma malha de teclas.

O *layout* QWERTY (Figura 1) foi desenvolvido por Christopher Latham Sholes, em 1878, e era destinado a máquinas de escrever. Seu desenvolvimento se justificou pela necessidade de um arranjo que causasse menos obstrução nos mecanismos dessas máquinas, diminuindo a velocidade de digitação. Dessa forma, esse *layout* consistiu no posicionamento de pares recorrentes de caracteres em lados opostos do teclado. Isso facilitou o uso das duas mãos e contribuiu para a popularização desse arranjo (ALSWAIDAN; HOSNY; NAJJAR, 2015; BI; SMITH; ZHAI, 2012; MACKENZIE; ZHANG; SOUKOREFF, 1999; YAMADA, 1980). Por causa de sua crescente popularidade, arranjos derivados do QWERTY foram surgindo com o tempo, como o QWERTZ, usado na Alemanha e outros países da Europa Central e o AZERTY, mais comumente utilizado em países de língua francesa na Europa e África. Posteriormente, o *layout* QWERTY foi transportado para os teclados dos computadores, facilitando a experiência de usuários já familiarizados com o arranjo das máquinas de escrever, mesmo que já não existissem os problemas relacionados à obstrução de mecanismos.

Algumas tentativas foram feitas com o objetivo de se desenvolverem novos *layouts* de teclados mais eficientes que o QWERTY. São os casos do Dvorak (Figura 2), criado por August Dvorak em 1982 e o Colemak (Figura 3), desenvolvido por Shai Coleman e lançado em 2006.

Esses novos *layouts* foram concebidos para teclados físicos e o custo de rearranjar

Figura 1: *Layout* QWERTY
Fonte: Wikipedia (adaptado)

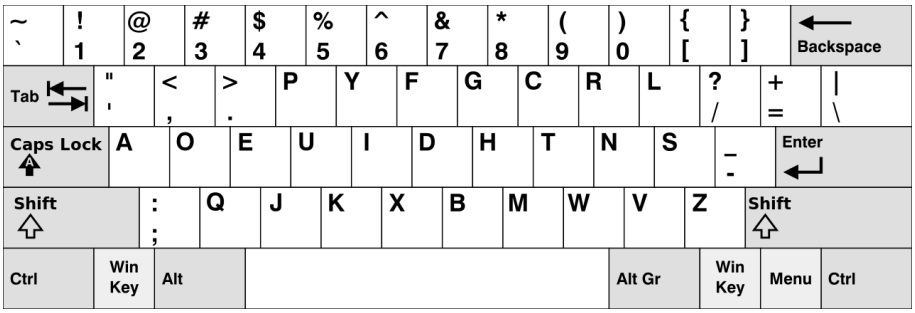


Figura 2: *Layout* Dvorak
Fonte: Wikipedia

caracteres, além do tempo necessário para aprender o novo posicionamento fizeram com que não fossem bem recebido pelos usuários, contribuindo para que o *layout* QWERTY permanecesse intocado em sua posição de popularidade.

Posteriormente, o desenvolvimento e difusão da tecnologia da tela sensível ao toque (*touchscreen*) em dispositivos eletrônicos portáteis como *tablets* e *smartphones*, e em terminais de autoatendimento, permitiu que o teclado físico fosse dispensado, em prol de um teclado virtual, que pode aparecer ou não na tela do aparelho, de acordo com a necessidade do usuário. Contando ainda com a familiaridade dos usuários com o QWERTY, este *layout* ainda vem sendo transportado para os novos aparelhos, mesmo sob novas condições de exibição. A Figura 4 mostra uma versão do *layout* QWERTY utilizada em *smartphones*.

Os teclados físicos dos computadores *desktop* e *laptops* são geralmente manipulados utilizando-se os 10 dedos das mãos, alternando seu uso. Quando se tratam de dispositivos com *touchscreen*, a entrada de dados em seus teclados virtuais geralmente ocorre por meio de dois polegares ou através de um dedo, polegar ou indicador.

O problema de otimização que busca definir a disposição dos caracteres nas teclas de um teclado para um dedo é conhecido como *single-finger keyboard layout problem* (SK-QAP) (DELL'AMICO et al., 2009). O SK-QAP possui natureza combinatória e, problemas com essa característica geralmente são resolvidos por meio de algoritmos heurísticos. O termo "heurística" tem raiz etimológica na palavra grega *heuriskein*, que significa encontrar

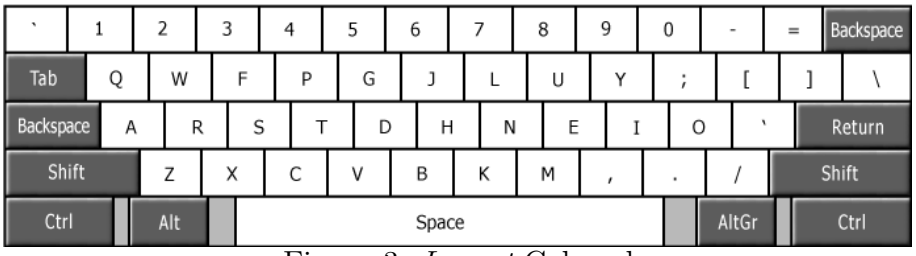


Figura 3: *Layout* Colemak
Fonte: Colemak keyboard layout

Figura 4: *Layout* QWERTY em teclado virtual
Fonte: Tecnoblog

ou descobrir. No âmbito da Pesquisa Operacional, heurísticas são métodos de tomada de decisão, que se baseiam em ideias simples para resolver um problema de otimização, com um esforço computacional razoável. Os métodos heurísticos são capazes de obter boas soluções viáveis, porém sem garantir sua proximidade da solução ótima. A classe de heurísticas pode ser dividida em heurísticas específicas e meta-heurísticas. As heurísticas específicas são desenvolvidas para atacar um problema ou instância específica. As meta-heurísticas, por outro lado, são de natureza mais geral, podendo ser aplicadas na resolução da maioria dos problemas de otimização (SOUZA, 2011; TALBI, 2009).

Este trabalho propõe um algoritmo heurístico para resolução do *single-finger keyboard layout problem*.

1.2 Justificativa

Ainda que novas tecnologias para entrada de dados venham surgindo recentemente no mercado, como comandos de voz, por exemplo, o teclado virtual ainda é o método mais popular para realizar ações dessa natureza em dispositivos móveis.

A empresa Kaspersky Lab conduziu, em 2017, uma pesquisa com 20082 usuários de dispositivos com conexão à internet para determinar quais atividades *online* eram as mais populares. Considerando dispositivos móveis, o uso de *e-mails* foi classificado como a atividade mais frequente no âmbito mundial, presente no dia a dia de 67% dos respondentes. Para o Brasil, 68% dos participantes afirmaram utilizar *e-mails* em *tablets*

e *smartphones* e, 66% dos portugueses tem esta atividade como prática constante. O uso de redes sociais e mensagens instantâneas também aparecem entre as 10 atividades *online* mais frequentes em *smartphones* e *tablets* nos principais países de língua portuguesa e no mundo. Essas informações são reforçada por pesquisa desenvolvida pela empresa Kantar IBOPE Media (2016), que concluiu que 88% dos brasileiros usam *smartphones* para troca de mensagens. Todas essas atividades envolvem entrada de texto, geralmente por meio de um teclado virtual associado a esses dispositivos. As Figuras 5 e 6 ilustram as dez atividades mais comuns em ambiente *online*, de acordo com a pesquisa citada, mundialmente e no Brasil, respectivamente.

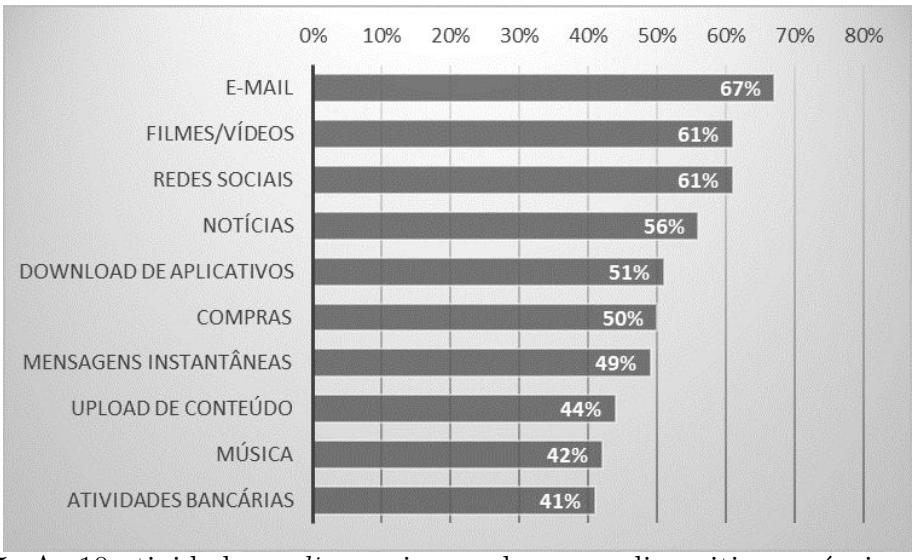


Figura 5: As 10 atividades *online* mais populares em dispositivos móveis no mundo
Fonte: (Kaspersky Lab, 2017)(Dados)

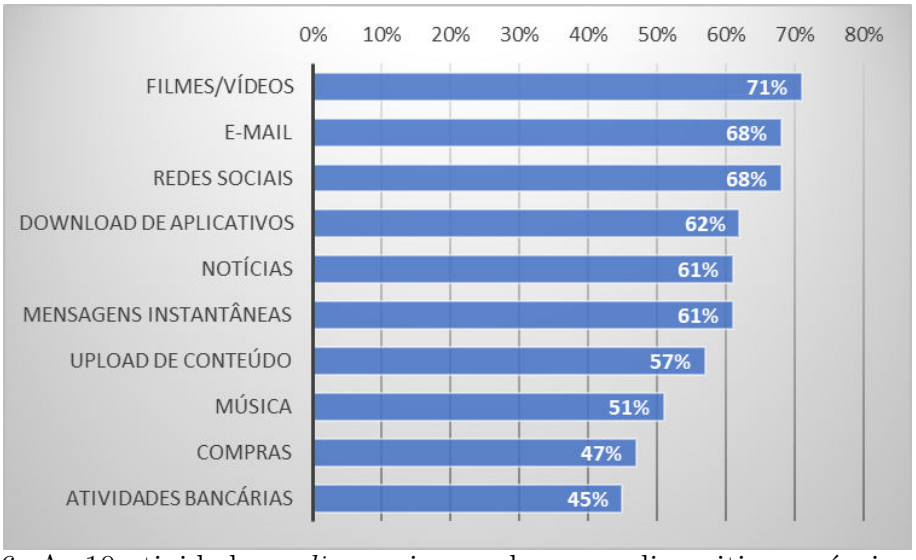


Figura 6: As 10 atividades *online* mais populares em dispositivos móveis no Brasil
Fonte: (Kaspersky Lab, 2017)(Dados)

Como já mencionado, o *layout* mais comumente utilizado em teclados virtuais,

hoje, é ainda o QWERTY, desenvolvido originalmente para máquinas de escrever. Esse arranjo é inadequado quando se considera o uso de um dedo para digitação, já que, como pares de caracteres frequentemente escritos em sequência são posicionados em lados opostos, o dedo deve passar constantemente de um lado para outro do teclado, o que gera uma quantidade considerável de movimentos desnecessários. Já não existem mais os problemas de obstrução dos mecanismos de máquinas de escrever ou mesmo a dificuldade de mudar teclas de lugar num teclado físico, já que os teclados virtuais se tratam de uma interface gráfica, sendo fácil o rearranjo de caracteres em suas teclas (ALSWAIDAN; HOSNY; NAJJAR, 2015; BI; SMITH; ZHAI, 2010, 2012; MACKENZIE; ZHANG, 1999; MACKENZIE; ZHANG; SOUKOREFF, 1999; RAYNAL; VIGOUROUX, 2005; SCHEDLBAUER, 2007). Dessa forma, um *layout* para teclados virtuais, destinados ao uso com um dedo, deve ser desenvolvido tendo em mente as particularidades desse estilo de digitação, tendendo-se a aproximar os caracteres cujos pares ocorrem com mais frequência, diminuindo o tempo de movimentação de uma tecla para outra durante a produção de um texto.

Apesar da clara necessidade de um *layout* de teclado mais eficiente para as atividades realizadas num dispositivo móvel ou mesmo em um terminal de autoatendimento, poucas pesquisas exploram este tema usando a abordagem da Pesquisa Operacional.

O problema de *layout* de teclado para um dedo (*single-finger keyboard layout problem*) é relevante para a Engenharia de Produção, já que possui interdisciplinaridade com várias de suas subáreas. Uma das relações mais imediatas é com o campo da Ergonomia, uma vez que existe a preocupação com a interação homem-máquina e a intenção do aumento da produtividade. Relaciona-se também com o campo da Engenharia de Produto já que trata da concepção de um novo produto baseado nas necessidades existentes no mercado de maior rapidez e eficiência nas comunicações através de texto. Possui relação com a área de Engenharia de Operações e Logística pois, ao otimizar um teclado em um terminal de autoatendimento, visa-se diminuir o tempo demandado para execução de operações, ocasionando filas menores. Há também, nessa mesma área, o paralelo com o planejamento de *layout* fabril, de maneira que, no último, existem facilidades a serem alocadas para que se obtenha o menor custo de comunicação entre elas, e, no SK-QAP, existem caracteres a serem alocados em teclas de maneira a minimizar a distância percorrida por um dedo na produção textual. Por fim, o problema de *layout* de teclado para um dedo tem relação com a Pesquisa Operacional, que pode ser considerada como uma das principais ferramentas destinada à resolução de problemas provenientes da Engenharia de Produção.

Em virtude da relação entre o SK-QAP e o *Quadratic Assignment Problem* (Problema Quadrático de Alocação, ou QAP), Dell'Amico et al. (2009) formularam o primeiro

como sendo uma variante do segundo. O QAP é um problema clássico e NP-difícil (SAHNI; GONZALEZ, 1976) conhecido por sua dificuldade de resolução, possuindo instâncias de tamanho limitado ainda não resolvidas de forma ótima. O SK-QAP é também considerado um problema NP-difícil (DELL'AMICO et al., 2009), uma vez que se trata de uma variante do QAP. Sabe-se que problemas dessa classe não possuem algoritmos capazes de determinar uma solução ótima em tempo polinomial. Como o SK-QAP é difícil de ser solucionado por métodos exatos, se faz necessária a utilização de métodos heurísticos para sua resolução.

Uma meta-heurística deve ser desenvolvida para ser simples e funcionar bem sem qualquer conhecimento prévio sobre o problema a ser resolvido. Para este trabalho, decidiu-se pela implementação de uma meta-heurística baseada em busca local, uma vez que se trata de uma abordagem simples e robusta capaz de obter soluções de alta qualidade em tempos computacionais razoáveis, sendo um método popular para resolução de problemas da classe NP-difícil (FOCACCI; LABURTHE; LODI, 2004; LOURENÇO; MARTIN; STÜTZLE, 2010; OSMAN; LAPORTE, 1996).

No trabalho de Dell'Amico et al. (2009) foram desenvolvidos teclados otimizados para um dedo para inglês, francês, italiano e espanhol. De acordo com dados do Ministério da Cultura (2017), o português é a quarta língua mais falada do mundo, com 260 milhões de falantes, sendo o Brasil responsável por 80% deste contingente. Com a significativa quantidade de usuários de dispositivos *touchscreen* que têm o português como sua primeira língua, existe a necessidade de um teclado otimizado para este idioma, uma lacuna nas pesquisas relacionadas ao desenvolvimento *layouts* para teclados para um dedo.

1.3 Objetivos

Os objetivos deste trabalho são expostos a seguir.

1.3.1 Objetivo geral

Propor uma meta-heurística baseada em busca local para o problema de *layout* de teclado para um dedo (*Single-finger Keyboard Layout Problem*).

1.3.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Identificar trabalhos relacionados.
- Desenvolver estratégia para gerar soluções iniciais.

- Implementar estruturas de vizinhança.
- Elaborar mecanismos de perturbação.
- Testar o algoritmo proposto nas instâncias existentes na literatura para os idiomas: espanhol, francês, inglês e italiano.
- Resolver o SK-QAP para a língua portuguesa

1.4 Estrutura do trabalho

O restante deste trabalho será estruturado como se segue: O Capítulo 2 contém uma revisão dos trabalhos relacionados ao SK-QAP e de algumas das meta-heurísticas encontradas na literatura para resolver o QAP, além de apresentar a formulação matemática do SK-QAP e trazer definições de conceitos relevantes. No Capítulo 3 um algoritmo baseado em busca local, para resolução do SK-QAP, é proposto, apresentando-se as estruturas de vizinhança e perturbações incorporadas ao procedimento. O conjunto de instâncias desenvolvido para a língua portuguesa também é apresentado nesse capítulo. O Capítulo 4 contém os resultados computacionais e o Capítulo 5 relata as conclusões do trabalho e sugestões para trabalhos futuros.

Capítulo 2

Fundamentação Teórica e Revisão da Literatura

Este capítulo aborda conceitos relevantes a respeito de meta-heurísticas baseadas em busca local e em busca populacional, bem como define algumas das técnicas mais utilizadas pertencentes a esses grupos. Posteriormente, define-se o SK-QAP, apresentando suas características e formulação matemática. Por fim, é feita uma revisão de trabalhos relacionados, considerando os problemas de *layout* de teclado para um dedo bem como para n dedos.

2.1 Meta-heurísticas

Os problemas de otimização combinatória são geralmente provenientes de situações reais que envolvem tomadas de decisões e integram conceitos dos campos de programação linear, teoria de algoritmos e análise combinatória, cujo objetivo é buscar, em um conjunto finito de arranjos de objetos, o melhor arranjo possível, ou seja, um arranjo ótimo (LAWLER, 1976; SCHRIJVER, 2005).

Para obtenção de solução para problemas de otimização combinatória é possível o uso de três tipos de métodos: métodos exatos, algoritmos aproximados ou métodos heurísticos. O primeiro deles fornece a solução ótima para o problema, porém, para problemas difíceis, pode se tornar computacionalmente inviável. O segundo tipo garante a qualidade da solução subótima obtida, pois é possível computar sua distância com relação ao valor ótimo. O terceiro e último método se baseia em ideias simples e fornece, geralmente, uma boa solução mas sem garantia de qualidade. Os métodos heurísticos são desenvolvidos conforme as especificidades de um problema em particular (GENDREAU; POTVIN, 2010a; OSMAN; LAPORTE, 1996; SOUZA, 2011).

Meta-heurísticas, por sua vez, são uma classe de métodos aproximativos destina-

dos à solução de complexos problemas de otimização, capazes de desenvolver abordagens heurísticas, de forma a escapar de ótimos locais, utilizando diferentes estratégias exploratórias aplicadas ao espaço de busca. Elas se dividem em meta-heurísticas baseadas em busca local e baseadas em busca populacional, de acordo com a abordagem utilizada para explorar o espaço de soluções. As meta-heurísticas são mais gerais que os métodos heurísticos, e por isso, possuem maior variedade de aplicações (GENDREAU; POTVIN, 2010a; OSMAN; LAPORTE, 1996; SOUZA, 2011; TALBI, 2009).

2.1.1 Meta-heurísticas baseadas em busca local

A busca local forma uma classe de heurísticas capazes de gerar uma solução aproximada em um tempo computacional razoável. Inicialmente, conta-se com uma solução inicial, obtida a partir de heurística construtiva, por exemplo, e, a cada iteração, o espaço de busca é explorado à procura de uma solução mais promissora. A busca perdura até que não se encontrem mais melhoras, terminando ao se obter um ótimo local ou por outro critério de parada previamente especificado. Alguns critérios de parada a serem utilizados podem ser um número predeterminado de iterações, de iterações sucessivas sem melhora no valor da função objetivo ou um certo tempo de CPU atingido (OSMAN; LAPORTE, 1996; SOUZA, 2011).

Meta-heurísticas baseadas em busca local utilizam estruturas de vizinhança para explorar o espaço de busca. Uma vizinhança é uma função que associa um conjunto de soluções a uma solução inicial, geradas através de um operador que realiza movimentos viáveis, modificando levemente esta solução (GENDREAU; POTVIN, 2010a; OSMAN; LAPORTE, 1996; TALBI, 2009).

Dentre as principais meta-heurísticas baseadas em busca local, utilizadas para resolução de problemas de *layout* de teclados pode-se citar: *simulated annealing* (SA), *tabu search* (TS), *variable neighborhood search* (VNS) e *iterated local search* (ILS).

2.1.1.1 *Simulated Annealing* (SA)

O método *simulated annealing*, proposto por Kirkpatrick (1984), tem fundamentação na mecânica estatística utilizando analogia com termodinâmica, num paralelo em que a energia associada a um estado, no campo da física, corresponde a um custo de solução no campo da otimização, ou seja, o congelamento de um sistema (estado de energia mínima) representa a solução ótima de um problema de otimização (KOTSIREAS, 2013; SOUZA, 2011).

O processo de SA começa com uma solução inicial e, a cada iteração, um vizinho dessa solução é gerado de maneira aleatória ou a partir de uma regra predefinida e,

então, testado para determinar se substituirá a solução atual. Supondo um problema de minimização, tem-se s_0 como a solução inicial e T_0 como a temperatura inicial do sistema, geralmente um valor alto. A cada solução vizinha candidata s' originada, é computado um Δ, que é a diferença entre os valores da função objetivo obtidos a partir das duas soluções. Se $\Delta < 0$, ocorre a substituição de s_0 por s'. Caso contrário, s' pode ser aceita com a probabilidade de $e^{-\Delta/T}$, em que T é o parâmetro de controle de temperatura, de valor inicial T_0 que diminui conforme uma taxa de resfriamento α, sendo $0 < \alpha < 1$. Para cada valor de T existe um número máximo de iterações pré-determinado (SA_{max}) que deve ser atingido de modo a ocorrer variação do parâmetro. O método termina quando T chega a 0 ou valor muito próximo (KOTSIREAS, 2013; MARTINS; RIBEIRO, 2006; NIKOLAEV; JACOBSON, 2010; OSMAN; LAPORTE, 1996; SOUZA, 2011).

Os parâmetros α, T_0 e SA_{max} devem ser cuidadosamente estimados, visto que a convergência do método depende de uma boa decisão sobre os mesmos. Para determinação de SA_{max}, geralmente, são levadas em consideração as dimensões do problema, já T_0 pode ser definido através simulação ou o custo de soluções. A respeito de α, é necessária adoção de valor que não acarrete resfriamentos muito bruscos ou muito lentos. No caso de resfriamentos bruscos, (T baixo e α alto) o método pode ficar preso em um ótimo local de má qualidade. Já para casos de resfriamento lento, o tempo computacional necessário para a convergência se torna impraticável (CRAMA; KOLEN; PESCH, 1995; SOUZA, 2011).

Por permitir "pioras" na solução atual, o SA é eficiente em escapar de possíveis ótimos locais. O método é vantajoso, também, por sua simplicidade e fácil implementação, sendo frequentemente utilizado como parte de outras meta-heurísticas (CRAMA; KOLEN; PESCH, 1995; NIKOLAEV; JACOBSON, 2010).

O funcionamento de um procedimento SA básico para um problema de minimização é descrito no Algoritmo 1.

2.1.1.2 *Tabu Search* (TS)

Proposto por Glover (1986), *tabu search*, ou busca tabu, é um método capaz de guiar deterministicamente a busca de soluções de maneira a escapar de possíveis ótimos locais.

A partir de uma solução inicial, o espaço de busca é explorado de forma que a solução vizinha que melhora, ou menos deteriora a função objetivo, é tomada como solução corrente, sendo sua vizinhança explorada na próxima iteração. Para reduzir a probabilidade de ocorrerem ciclos, ou seja, que algoritmo fique preso em determinada região do espaço de busca, a TS conta com uma lista tabu, ou seja, uma memória de curto prazo (ou atributiva) capaz de registrar movimentos realizados recentemente (movimentações

Algoritmo 1 *Simulated Annealing*

Procedimento SA($f(.)$, $N(.)$, α, SA_{max}, T_0, s)
  $s^* \leftarrow s$ ▷ Melhor solução atual
  $ItT \leftarrow 0$ ▷ Número de iterações executadas em uma temperatura T
  $T \leftarrow T_0 \geq 0$ ▷ Temperatura inicial
  enquanto $T > 0$ **faça**
    enquanto $ItT < SA_{max}$ **faça**
      $ItT \leftarrow ItT + 1$
      Gere solução $s' \in N(s)$
      Calcule $\Delta_{s',s} = f(s') - f(s)$
      se $\Delta_{s',s} < 0$ **então**
        $s \leftarrow s'$
      senão
        $s \leftarrow s'$ com probabilidade de $e^{-\Delta/T}$
        se $f(s') < f(s^*)$ **então**
          $s^* \leftarrow s'$
    $T \leftarrow \alpha \times T$
    $ItT \leftarrow 0$
  retorne s^*
fim Procedimento

Fonte: Adaptado de: Kirkpatrick (1984), Nikolaev e Jacobson (2010), Souza (2011)

tabus), os quais são tidos como proibidos, evitando possível retorno a soluções previamente visitadas. Uma movimentação tabu pode ser realizada, no entanto, se retornar uma solução melhor do que a mais promissora encontrada até o momento. A busca tabu pode contar também com uma memória de longo prazo (ou explícita) que é responsável por armazenar componentes das melhores soluções encontradas (CRAMA; KOLEN; PESCH, 1995; GENDREAU; POTVIN, 2010b; GLOVER; LAGUNA, 1999; MARTINS; RIBEIRO, 2006; OSMAN; LAPORTE, 1996).

O algoritmo pode ser interrompido a partir de três critérios: Quando um número máximo de iterações pré-definido é atingido sem que haja melhora no valor da função objetivo; quando o melhor valor encontrado chega nas proximidades ou exatamente em um limite conhecido ou quando um número total de iterações ou determinado tempo de CPU é atingido (GENDREAU; POTVIN, 2010b; MARTINS; RIBEIRO, 2006; SOUZA, 2011).

Algumas estratégias podem ser incorporadas à TS de maneira a melhorar sua eficiência. A primeira delas é a função de aspiração que é capaz de retirar um movimento da lista tabu se o mesmo gerar uma solução não visitada que leve a função objetivo a um valor melhor que o nível de aspiração do valor atual. A segunda estratégia é chamada de intensificação e usa uma memória capaz de identificar características promissoras das melhores soluções visitadas de maneira a realizar buscas mais intensivas nas regiões do espaço de busca onde elas se encontram. A terceira estratégia é a diversificação e, ao contrário da intensificação, utiliza memória para forçar a busca na direção de regiões do espaço de soluções que não foram exploradas o suficiente. Outras estratégias, como uso de uma lista de candidatos ou de uma tabela *hash*, podem ser também introduzidas ao

método de forma a acelerar e filtrar a busca (CRAMA; KOLEN; PESCH, 1995; GENDREAU; POTVIN, 2010b; GLOVER; LAGUNA, 1999; MARTINS; RIBEIRO, 2006; SOUZA, 2011).

O procedimento básico da busca tabu é ilustrado no Algoritmo 2.

Algoritmo 2 *Tabu Search*

Procedimento TS($f(.)$, $N(.)$, $A(.)$, $|V|$, f_{min}, $|T|$, BT_{max}, s)
  $s^* \leftarrow s$ ▷ Melhor solução atual
  $It \leftarrow 0$ ▷ Número de Iterações
  $BestIt \leftarrow 0$ ▷ Iteração que levou à melhor solução
  $T \leftarrow \varnothing$ ▷ Lista Tabu
  Inicialize a função de aspiração A
  enquanto $f(s) > f_{min}$ e $It - BestIt \leq BT_{max}$ **faça**
    $It \leftarrow It + 1$
    Encontre melhor solução vizinha viável s' cujo movimento não pertença a T
    $s \leftarrow s'$
    Atualize T
    se $f(s) < f(s^*)$ **então**
      $s^* \leftarrow s$
      $BestIt \leftarrow It$
    Atualize A ▷ Função de aspiração
  retorne s^*
fim Procedimento

Fonte: Adaptado de: Glover (1986), Souza (2011)

2.1.1.3 *Variable Neighborhood Search* (VNS)

O método VNS foi proposto por Mladenović e Hansen (1997) e consiste em uma troca sistemática de vizinhanças em uma fase de descida em direção a um ótimo local.

Considera-se um conjunto de vizinhanças N_κ, com κ variando de 1 a κ_{max}. O procedimento começa com uma solução inicial s_0. Em seguida, atribui-se o valor de 1 para κ e, enquanto $\kappa \leq \kappa_{max}$ gera-se uma solução s', aleatoriamente, a partir da vizinhança N_κ da solução inicial. A seguir, a solução s' passa então por um processo de busca local, gerando s''. Se s'' proporcionar melhor valor para a função objetivo, ela será a nova solução inicial na próxima iteração e será atribuído o valor de 1 para κ. Caso contrário, κ é incrementado de 1. O método VNS se encerra quando atinge certa condição que pode ser: tempo máximo de funcionamento de CPU, máximo número de iterações totais, máximo número de iterações entre duas melhoras da função objetivo, ou o máximo número de vezes em que a vizinhança de maior ordem (N_κ) é atingida (HANSEN et al., 2010; MARTINS; RIBEIRO, 2006; MLADENOVIĆ; HANSEN, 1997; SOUZA, 2011).

O método descrito anteriormente é definido como VNS básico, no entanto, ao mesmo podem ser adicionadas extensões, como no caso do *general variable neighborhood search*, o que usa o processo *variable neighborhood descent* (VND) como estratégia de busca local, *skewed variable neighborhood search* (SVNS), que usa uma função $\rho(s, s'')$ para avaliar a distância entre uma solução atual s e um ótimo local s'', permitindo explorar

"vales" mais distantes da melhor solução atual ou o *variable neighborhood decomposition search* (VNDS) que decompõe o problema, aplicando, inicialmente, a busca local em uma variável (ou conjunto de variáveis), fixando-se as outras para, posteriormente, seguir com o VNS usual (HANSEN et al., 2010; MACEDO et al., 2015; SOUZA, 2011).

O pseudocódigo do procedimento VNS básico é ilustrado no Algoritmo 3.

Algoritmo 3 *Variable Neighborhood Search*

```
1: Procedimento VNS
2:     s* ← s_0                                   ▷ Gerar a solução inicial
3:     Defina κ_max                               ▷ Número de estruturas de vizinhança
4:     enquanto Critério de parada não for atingido faça
5:         κ ← 1                                  ▷ Estrutura de vizinhança atual
6:         enquanto κ ≤ κ_max faça
7:             Gere solução s' ∈ N_κ(s*)
8:             s'' ← BuscaLocal(s')
9:             se f(s'') < f(s*) então
10:                s* ← s''
11:                κ ← 1
12:            senão
13:                κ ← κ + 1
14:    retorne s*
15: fim Procedimento
```

Fonte: Adaptado de: Mladenović e Hansen (1997), Souza (2011)

2.1.1.4 *Iterated Local Search* (ILS)

O ILS é um método simples que parte do princípio que, a partir de uma solução inicial, uma busca local é realizada e, ao atingir valor ótimo local, um mecanismo de perturbação é ativado, de modo que a solução seja modificada a uma certa extensão de maneira a acessar vizinhança mais promissora em busca de novo ótimo local, dado um critério de aceitação. Dessa forma, é possível dizer que o método foca apenas nas soluções retornadas por uma heurística de busca local ao invés de todo o espaço de busca. O enviesamento na escolha de soluções é considerado o ponto forte deste procedimento (BLUM; ROLI, 2003; LOURENÇO; MARTIN; STÜTZLE, 2010; SOUZA, 2011).

Para desenvolvimento de um algoritmo ILS é preciso definir quatro de seus componentes: mecanismo de geração de solução inicial, processo de busca local, procedimento de perturbação e critério de aceitação. A solução inicial pode ser obtida de diversas formas, sendo a aleatoriedade um bom ponto de partida, pois garante maior diversificação no início do método. O processo de busca local é, por sua vez, fundamental para definir o desempenho do ILS em termos de velocidade de execução e qualidade da solução. A técnica de perturbação se mostra como uma escolha delicada já que contribui para o procedimento de diversificação do algoritmo. A escolha de uma perturbação pequena pode ocasionar retorno ao ótimo local previamente visitado quando a solução perturbada

passar pelo processo de busca local. Se, por outro lado, a técnica escolhida gerar uma perturbação muito grande, será como iniciar de um novo ponto de partida do algoritmo, sem guardar características de uma solução promissora já examinada. É recomendado o uso de perturbações aleatórias ou adaptativas, de maneira a evitar ciclos do algoritmo. O critério de aceitação é o componente que contribui para o processo de intensificação do algoritmo. Algumas abordagens usadas são aceitar um novo ótimo local se houver melhora no valor da função objetivo ou sempre aceitar a nova solução (BLUM; ROLI, 2003; LOURENÇO; MARTIN; STÜTZLE, 2010; SOUZA, 2011).

Uma versão básica do ILS é mostrada no Algoritmo 4.

Algoritmo 4 *Iterated Local Search*

```
1: Procedimento ILS
2:     s_0 ← GerarSoluçãoInicial()
3:     s* ← BuscaLocal(s_0)
4:     enquanto Critério de parada não for atingido faça
5:         s' ← Perturbacao(s, histórico de busca)
6:         s'' ← BuscaLocal(s')
7:         s* ← Aceitacao(s*, s'', histórico de busca)
8: fim Procedimento
```

Fonte: Adaptado de: Lourenço, Martin e Stützle (2010), Souza (2011)

2.1.2 Meta-heurísticas baseadas em busca populacional

Métodos baseados em busca populacional consistem na definição de uma população inicial de possíveis soluções, na qual é realizada uma busca a partir de múltiplos pontos iniciais, paralelamente. Métodos de busca populacional têm grande capacidade de exploração já que diversas boas soluções são mantidas simultaneamente, sendo combinadas de forma a produzir resultados melhores, porém sua performance depende do modo como a população é manipulada. O uso de uma população permite que uma técnica de busca identifique e foque em regiões mais promissoras do espaço, explorando propriedades comuns a boas soluções (BLUM; ROLI, 2003; BEHESHTI; SHAMSUDDIN, 2013; BOLUFÉ-RÖHLER; CHEN, 2013; CRAMA; KOLEN; PESCH, 1995; SOUZA, 2011).

Dentre as principais meta-heurísticas baseadas em busca populacional, utilizadas para resolução de problemas de *layout* de teclados pode-se citar *Genetic Algorithms* (GA), *Ant Colony Optimization* (ACO) e *Particle Swarm Optimization* (PSO).

2.1.2.1 *Genetic Algorithms* (GA)

Meta-heurística inicialmente proposta por Holland (1992), os *genetic algorithms* (ou algoritmos genéticos) se baseiam na adaptação de processos evolutivos, encontrados na natureza, para problemas de otimização. Nesse método, indivíduos mais aptos de

uma população possuem mais chances de sobrevivência e de produzir proles mais aptas, ao passo que, aqueles com características menos favoráveis, tendem ao desaparecimento. Nos GA's, uma população é um conjunto de cromossomos (soluções) que, por sua vez, contêm genes (componentes da solução). Os alelos, nessa alusão aos processos evolutivos, são os valores que cada componente da solução pode assumir. As soluções têm aptidão avaliada, geralmente, pelos valores da função objetivo obtidos (BURKARD, 2013; CRAMA; KOLEN; PESCH, 1995; KOTSIREAS, 2013; MARTINS; RIBEIRO, 2006; OSMAN; LAPORTE, 1996; REEVES, 2010; SOUZA, 2011).

O método GA começa com uma população inicial gerada com tamanho tal que possa dar condições ao algoritmo explorar o máximo possível de regiões do espaço de busca. Uma população inicial de tamanho pequeno prejudica a diversificação do algoritmo, fazendo com que o espaço de busca não seja explorado de maneira efetiva. Uma população de tamanho elevado, porém, pode demandar longo tempo de CPU no processo de determinação de uma solução viável. A população inicial tem sua aptidão avaliada e, posteriormente, sofre a ação de operadores como o *crossover* e a mutação, contando-se as gerações passadas a cada iteração. O operador *crossover* funciona selecionando dois indivíduos da população e recombinando seus genes para gerar uma descendência. O operador mutação, no entanto, trabalha modificando alelos do indivíduo, de modo a transformá-lo em um novo cromossomo. Geralmente são usadas probabilidades associadas aos operadores, sendo mais comum a adoção de probabilidades mais elevadas para o operador *crossover* (CRAMA; KOLEN; PESCH, 1995; OSMAN; LAPORTE, 1996; REEVES, 2010; SOUZA, 2011).

Geralmente é utilizado um de três critérios para determinar sobrevivência dos indivíduos: i) aleatório; ii) roleta, no qual indivíduos mais aptos têm mais chances de sobreviver; iii) combinação dos critérios anteriores. Esses critérios garantem alguma sobrevivência de indivíduos menos qualificados, o que garante maior diversificação da população e, portanto, maior chance de evadir-se de ótimos locais. Outras estratégias para manter a diversificação são o uso de populações maiores e de taxas de mutação mais frequentes (CRAMA; KOLEN; PESCH, 1995; REEVES, 2010; SOUZA, 2011).

O método termina quando atinge algum critério de parada. Os critérios mais comuns são parar quando se alcança certo número de gerações, quando não são obtidas melhoras na função objetivo após certo número de iterações ou quando se chega a certo valor de desvio padrão da população ou a um limite de tempo de execução (BURKARD, 2013; KOTSIREAS, 2013; MARTINS; RIBEIRO, 2006; REEVES, 2010; SOUZA, 2011).

O pseudocódigo básico de um algoritmo genético é mostrado no Algoritmo 5

Algoritmo 5 *Genetic Algorithm*

```
1: Procedimento GA
2:     Gere população de p indivíduos                          ▷ Gerar população inicial
3:     Avalie os p indivíduos
4:     enquanto Critério de parada não for atingido faça
5:         Gere f filhos a partir de p pais                    ▷ Nova população
6:         Avalie os f filhos
7:         Selecione a população sobrevivente de p indivíduos a partir de pais e filhos
8: fim Procedimento
```

Fonte: Adaptado de: Holland (1992), Talbi (2009), Souza (2011)

2.1.2.2 *Ant Colony Optimization* (ACO)

A meta-heurística *ant colony optimization* (ou colônia de formigas) foi introduzida no campo da otimização através do trabalho de Dorigo e Stützle (2010) e simula a movimentação de um conjunto formigas em busca de alimento. Esses animais procuram o caminho mais curto até a fonte de alimento, deixando uma trilha de feromônios guiando outras formigas, da colônia à comida (BLUM; ROLI, 2003; BURKARD, 2013; DORIGO; STÜTZLE, 2010; SOUZA, 2011).

A analogia com a otimização combinatória se faz através de que a área percorrida pelas formigas, a quantidade de alimento nas fontes e a trilha de feromônios se traduzem como um conjunto de soluções viáveis para um problema de otimização, o valor da função objetivo e um componente de memória adaptativa, respectivamente. As formigas de um ACO são procedimentos estocásticos para construir caminhos no espaço de busca, componente a componente, usando informações específicas da instância abordada e trilhas de feromônios que refletem experiências adquiridas durante o processo de busca (BLUM; ROLI, 2003; BURKARD, 2013; DORIGO; STÜTZLE, 2010; SOUZA, 2011).

O procedimento começa com a inicialização de parâmetros e trilhas de feromônios. O laço principal que se segue é composto por três passos: no primeiro deles, um conjunto de formigas constrói uma solução passo a passo, adicionando componentes viáveis a uma solução anteriormente vazia. Em seguida, tem-se a opção de aplicar busca local à solução construída. O terceiro passo do laço é caracterizado pela adaptação das trilhas de feromônios. Esse passo garante que formigas atuantes nas próximas iterações possam ser atraídas a elementos de soluções mais promissoras. A atualização dessa trilha pode se dar por meio de depósito de feromônios em componentes de boas soluções ou por evaporação da substância, que diminui sua intensidade com o passar do tempo (BLUM; ROLI, 2003; DORIGO; STÜTZLE, 2010; SOUZA, 2011).

Um modelo básico para o procedimento ACO é mostrado no Algoritmo 6.

Algoritmo 6 *Ant Colony Optimization*

```
1: Procedimento ACO
2:     Inicialize a trilha de feromônios
3:     enquanto Critério de parada não for atingido faça
4:         para cada formiga m = 1, ..., m_max faça
5:             Construa solução usando a trilha de feromônios
6:             Atualize a trilha de feromônios (depósito ou evaporação)
7: fim Procedimento
```

Fonte: Adaptado de: Dorigo e Stützle (2010), Talbi (2009)

2.1.2.3 *Particle Swarm Optimization* (PSO)

Proposto inicialmente por Kennedy e Eberhart (1995), o algoritmo PSO, ou otimização por enxame de partículas, se inspira em eventos naturais relacionados ao comportamento social de alguns organismos, como bandos de pássaros ou cardumes de peixes, em busca de comida. O PSO se diferencia do GA por não haver um processo de seleção, já que, ao final do procedimento, a população inteira sobrevive. (KENNEDY, 2010; POLI; KENNEDY; BLACKWELL, 2007; TALBI, 2009).

Nesse método, cada partícula representa uma solução candidata ao problema abordado. Essas entidades carregam dados de velocidade (ou tamanho do passo) (v_i) e posição (x_i) e o processo de otimização nesse método ocorre por meio da interação entre estas entidades, em uma rede de comunicações com um número pré-estabelecido de partículas vizinhas. Nesse caso, a velocidade representa o tamanho da mudança que sofrerá a partícula i a cada iteração. (KENNEDY, 2010; POLI; KENNEDY; BLACKWELL, 2007; TALBI, 2009).

O procedimento ocorre com a avaliação de uma função $f(x)$ para cada partícula i, considerando sua posição x_i e velocidade v_i. Este resultado é comparado com a melhor posição atingida pela partícula ($pbest_i$), ou a melhor posição alcançada pelo conjunto (ou por um subconjunto) de partículas ($gbest$), guardando suas melhores coordenadas no espaço de busca a cada solução mais promissora encontrada. Com o passar das iterações, a tendência é que as partículas se juntem ao redor de ótimos locais, num comportamento semelhante a enxames (EBERHART; KENNEDY, 1995; KENNEDY, 2010; POLI; KENNEDY; BLACKWELL, 2007; TALBI, 2009).

Um pseudocódigo de modelo do PSO está descrito no Algoritmo 7.

2.2 O problema de *layout* de teclado para um dedo

2.2.1 Definição do problema e formulação matemática

O problema associado ao desenvolvimento de *layout* de teclado para um dedo é uma variante do problema de otimização combinatória nomeado de *quadratic assignment*

Algoritmo 7 *Particle Swarm Optimization*

```
1: Procedimento PSO
2:     Inicialize o enxame aleatoriamente;
3:     enquanto Critério de parada não for atingido faça
4:         Avalie f(x_i);
5:         para todas as partículas i faça
6:             Atualize velocidades v_i;
7:             Move para nova posição x_i;
8:             se f(x_i) < f(pbest_i) então
9:                 pbest_i = x_i;
10:            se f(x_i) < f(gbest) então
11:                gbest = x_i;
12:            Atualize (x_i, v_i);
13: fim Procedimento
```

Fonte: Adaptado de: Kennedy e Eberhart (1995), Talbi (2009)

problem (QAP) ou problema quadrático de alocação. Esse problema foi apresentado por Koopmans e Beckmann (1957), e pode ser definido como o problema relacionado à alocação de facilidades a locais, desta forma: Dado um conjunto de N facilidades a serem alocadas a N locais, com uma distância associada d_{kl} para cada par de locais k e l e um fluxo f_{ij} associado a cada par de facilidades i e j, tem-se, como objetivo, a minimização do produto total fluxo-distância, com as restrições de que uma facilidade só pode ser alocada a um local e cada local só pode receber uma facilidade (BURKARD; RENDL, 1984; BURKARD; KARISCH; RENDL, 1997; GAMBARDELLA; TAILLARD; DORIGO, 1999; NUGENT; VOLLMANN; RUML, 1968).

No caso do SK-QAP, há um conjunto $C = 1, 2, ..., n$ de caracteres a serem alocados a um conjunto $M = 1, 2, ..., e = p \times q$ de teclas (locais), com p e q suficientemente grandes para comportar os n caracteres em sua totalidade, de maneira que cada caractere seja designado a uma tecla e cada tecla receba, no máximo, um caractere. Todas as teclas são iguais e de mesmo tamanho, de formato retangular, distribuídas em uma rede de dimensões e, contendo caracteres diferentes em todas elas. Existe, também, uma matriz F de fluxos f_{ij} entre pares de caracteres i e j e uma matriz D contendo valor da função de Fitts d_{kl} para as teclas k e l. Como variáveis de decisão têm-se x_{ik} e x_{jl} que são variáveis binárias. A variável x_{ik} assume valor 1 quando o caractere i é alocado na tecla k, e 0, caso contrário. O mesmo é verdade para a variável x_{jl}, o caractere j e a tecla l.

A lei de Fitts (1954) leva em consideração tempo e dificuldades durante a realização de um movimento para uma determinada área, de maneira que o tempo necessário para mover-se (TM) de um ponto inicial a um ponto final é proporcional ao índice de dificuldade (ID). Existem algumas formulações da lei de Fitts, porém aquela mostrada na Equação (1) é mais condizente com dados empíricos (MACKENZIE; SELLEN; BUXTON, 1991) e, portanto, mais comumente utilizada nas pesquisas relacionadas ao *single-finger*

keyboard layout problem.

$$TM = a + b * log_2\left(\frac{S}{A} + 1\right) \tag{1}$$

Nessa formulação, a e b são constantes de valores pré-fixados, S é a distância entre os centros de duas teclas e A, a área de cada tecla (ALSWAIDAN; HOSNY; NAJJAR, 2015; DELL'AMICO et al., 2009; FITTS, 1954; LI; CHEN; GOONETILLEKE, 2006; MACKENZIE; SELLEN; BUXTON, 1991; MACKENZIE; ZHANG, 1999; ZHAI; HUNTER; SMITH, 2002). A lei de Fitts (Equação (1)) não leva em consideração o tempo de reação de uma pessoa ao digitar, considerando-se, portanto, que o digitador é experiente e tem conhecimento do *layout* do teclado utilizado. O SK-QAP pode ser formulado, então, da seguinte maneira:

$$z = min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{e} \sum_{l=1}^{e} f_{ij} d_{kl} x_{ik} x_{jl} \tag{2}$$

Sujeito a:

$$\sum_{k=1}^{e} x_{ik} = 1 \qquad i \in C \tag{3}$$

$$\sum_{i=1}^{n} x_{ik} \leq 1 \qquad k \in M \tag{4}$$

$$x_{ik} \in \{0, 1\} \qquad i \in C, k \in M. \tag{5}$$

A função objetivo (2) minimiza o produto do fluxo entre os caracteres i e j pelo valor da função de Fitts (Equação (1)) entre as teclas k e l pelas variáveis de decisão x_{ik} e x_{jl}. As restrições (3) definem que cada caractere precisa, necessariamente, estar alocado a uma tecla. As restrições (4) especificam que nem toda tecla precisa ter um caractere associado a ela, porém um caractere é o máximo que pode ser alocado nesse espaço. Já o conjunto de restrições (5) define os possíveis valores assumidos pelas variáveis de decisão.

2.2.2 Trabalhos relacionados

O SK-QAP é um problema relativamente novo na literatura, se comparado a problemas clássicos, e portanto ainda não foi amplamente debatido, especialmente considerando abordagens inseridas na Pesquisa Operacional. Sendo assim, existe certa dificuldade em reunir literatura vasta sobre o mesmo, de forma que, para obter sólido embasamento para desenvolver esse estudo, foi necessário expandir o escopo dos trabalhos considerados a fazer parte dessa sessão. Além dos estudos que abordam a otimização para teclados de um dedo, foram incorporados aqueles que abordam melhoria de *layouts* para teclados de n

dedos (*n-finger keyboards*), onde n pode ser dois (polegares) ou todos os dedos das mãos.

2.2.2.1 Teclados para um dedo

As pesquisas em otimização para teclados de um dedo iniciaram-se com a necessidade de melhorar as performances de digitação em teclados virtuais que devem ser manipulados usando um dedo ou uma caneta *stylus*. As melhoras mais frequentemente buscadas nas pesquisas são maior velocidade de digitação e menor taxa de erros durante a produção de um texto.

MacKenzie e Zhang (1999) desenvolveram um modelo preditivo para determinar os limites superiores, medidos em palavras por minuto (ppm), de *layouts* de teclado. Esse modelo utilizou a lei de Fitts (Equação (1)) para computar o tempo de digitação de todos os pares de caracteres (dígrafos), ponderando os tempos de acordo com a ocorrência desses pares na língua inglesa. O *layout* QWERTY foi avaliado em 43,2 ppm. Os autores, então, se valeram da tentativa e erro para chegar a um arranjo de caracteres cujo limite superior previsto pelo modelo desenvolvido fosse maior que aquele determinado para o QWERTY. O resultado escolhido foi denominado OPTI e seu limite superior avaliado em 58,2 ppm, 35% mais rápido que o QWERTY. Cinco pessoas participaram de um experimento durante 20 sessões de 45 minutos para avaliar o novo *layout* proposto. O *layout* OPTI é mostrado na Figura 7.

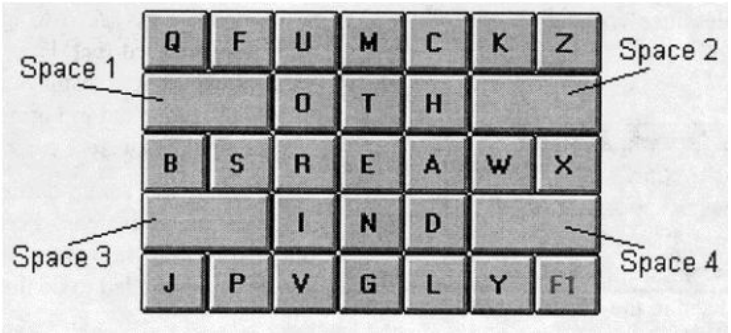


Figura 7: Teclado OPTI
Fonte: MacKenzie e Zhang (1999), p. 3

O trabalho de Zhai, Hunter e Smith (2002) calculou uma estimativa de palavras por minuto em 6 *layouts* diferentes, incluindo o QWERTY e o OPTI de MacKenzie e Zhang (1999). Para determinar se seria possível um teclado virtual mais eficiente que os avaliados, os autores utilizaram técnicas quantitativas baseadas na física para desenvolver *layouts* alternativos. A primeira técnica foi chamada de *dinamic simulation method* e trata de uma simulação que considera a existência de molas passando por cada par de teclas, cuja elasticidade depende da frequência entre as letras alocadas. Para escapar de mínimos locais, foram desenvolvidas duas estratégias. A primeira delas foi a experimentação com

diversos estados iniciais. A segunda foi a adição de um suporte a cada mola, fazendo com que as mesmas se separassem e a passagem de outras teclas por esses espaços fosse permitida, de forma a atingir melhores *layouts*. Iterações foram realizadas até que um arranjo satisfatório fosse encontrado. O teclado derivado dessa técnica foi chamado teclado Hooke e está ilustrado na Figura 8. O nome desse *layout* foi baseado na Lei de Hooke, relacionada à elasticidade dos corpos.

Figura 8: Teclado Hooke
Fonte: Zhai, Hunter e Smith (2002), p. 19

A segunda técnica usada por Zhai, Hunter e Smith (2002) iniciou-se com o desenvolvimento de um *software* em que, a cada passo, uma tecla escolhida aleatoriamente é movida a uma direção e valor aleatórios. A configuração é então avaliada, em termos de palavras por minuto, a partir de uma equação denominada *Fitts-digraph energy*. Essa equação consiste no produto entre dois termos, somado para todos os 27x27 dígrafos da língua inglesa. O primeiro termo é a frequência de dígrafos, calculada como a razão entre o número de ocorrências de um certo par de caracteres pelo número total de dígrafos da língua. O segundo termo é a função de Fitts (Equação (1)),com o parâmetro a igual a 0 e o parâmetro b igual a $1/4,9$.

Os autores usaram um algoritmo Metropolis (METROPOLIS et al., 1953) para determinar a probabilidade de uma nova configuração ser mantida como solução inicial numa próxima iteração. A função relaciona variações de energia e uma temperatura, que passa por um processo de *annealing* iterativo em ciclos de aumento e diminuição até não haver mais melhoras na solução. Os teclados gerados por esse método foram denominados teclados Metropolis (Figura 9).

Para melhorar a familiaridade dos usuários com o *layout* novo, os autores também desenvolveram uma técnica de refinamento alfabético, reposicionando caracteres do *layout* Metropolis conforme a ordem que aparecem no alfabeto. Os teclados desenvolvidos a partir dessa técnica foram denominados ATOMIK (*alphabetically tuned and optimized mobile interface keyboard*). Outro critério também utilizado nessa pesquisa foi o uso de

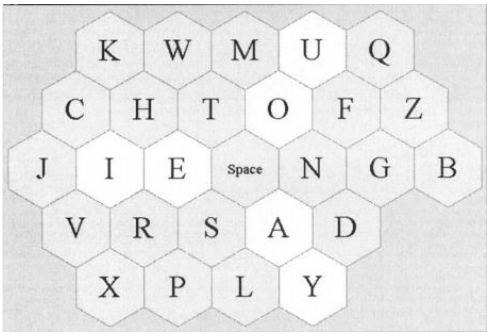


Figura 9: Teclado Metropolis
Fonte: Zhai, Hunter e Smith (2002), p. 23

um índice de conectividade de palavras, o qual foi computado para as 17 palavras mais frequentes na língua inglesa. Seu cálculo se dá através do produto entre a porcentagem de frequência da *i-ésima* palavra mais frequente pela pontuação de conectividade daquela palavra. A pontuação de conectividade depende da quantidade de dígrafos em sequência que se encontram adjacentes na configuração do teclado.

A Figura 10 mostra um teclado ATOMIK que satisfaz os 3 critérios desenvolvidos por Zhai, Hunter e Smith (2002).

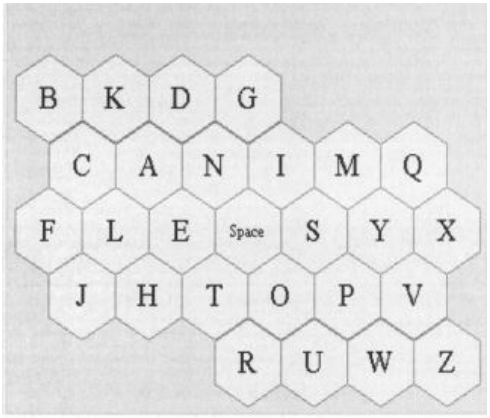


Figura 10: Teclado ATOMIK
Fonte: Zhai, Hunter e Smith (2002), p. 27

Raynal e Vigouroux (2005) desenvolveram um GA para ser usado no processo de otimização de qualquer teclado para qualquer língua. O processo começa com o posicionamento aleatório de caracteres num arranjo de teclas predefinido. As iterações do algortimo acabaram quando os melhores indivíduos estáveis foram encontradas. Um indivíduo foi considerado estável se manteve o mesmo arranjo de caracteres após 500 iterações. Os *layouts* desenvolvidos com essa técnica foram avaliados conforme metodologia apresentada no trabalho de SOUKOREFF e MACKENZIE (1995). Esse método produziu dois *layouts* para a língua inglesa nos formatos retangular e hexagonal (GAG I e GAG II), ambos com melhoras de mais de 50% em comparação com o QWERTY.

Li, Chen e Goonetilleke (2006) desenvolveram uma pesquisa para otimizar o tempo

de movimentação entre teclas, apenas para a língua inglesa. Os autores utilizaram o método empregado por MacKenzie e Zhang (1999) e Zhai, Hunter e Smith (2002) para avaliar os *layouts*. Para obter as frequências entre caracteres, os autores utilizaram as 15.000 palavras mais comuns da língua inglesa, de acordo com o *British National Corpus* (BNC), computando um espaço antes e depois de cada palavra. A abordagem do problema foi feita através de um modelo de programação inteira e uma heurística de dois estágios para resolvê-lo.

O modelo de programação inteira tem o custo de movimentação entre quaisquer duas teclas fixado. No momento seguinte, os autores introduzem o segundo modelo, no qual o primeiro estágio consiste em uma busca exata, em que uma nova solução é obtida a partir da troca entre dois caracteres, de maneira gulosa, até que a solução não atinja mais melhora após 10 tentativas. O segundo estágio, então, trata da aplicação da meta-heurística SA, para aumentar as chances de atingir um ótimo global. A Figura 11 ilustra os *layouts* desenvolvidos pelos autores. O melhor *layout* avaliado pelos autores foi o II, denominado YLAROF.

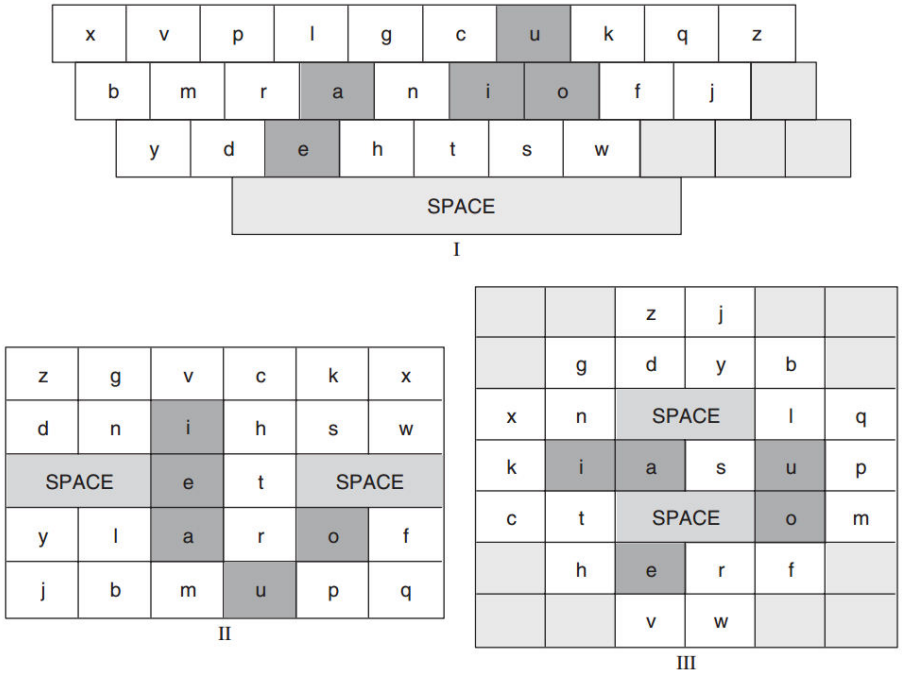


Figura 11: Arranjos otimizados para a língua inglesa
Fonte: Li, Chen e Goonetilleke (2006), p. 7

Dell'Amico et al. (2009) formalmente apresentaram o SK-QAP e desenvolveram *layouts* de teclados otimizados para um dedo para os idiomas inglês, espanhol, francês e italiano. O principal objetivo dos *layouts* propostos nesse trabalho foi minimizar o tempo gasto para escrever um texto. Para atingir esse propósito, os autores valeram-se das particularidades de cada língua abordada para alocação de caracteres, usando um corpus

específico para cada uma de maneira a determinar as frequências de dígrafos. Para cada língua foi gerado um conjunto de 6 instâncias, sendo uma delas criada através de dados reais, e, o restante, através de procedimento aleatório.

Três estruturas de vizinhança foram usadas no trabalho, chamadas de *contour filling* (N_1), *pairwise exchange* (N_2) e *k-exchange* (N_k), além de duas técnicas de aceleração (*speed up*). A estrutura de vizinhança N_1 é definida como o conjunto de soluções obtidas através da movimentação de um caractere para um local de contorno. Locais de contorno são um conjunto de teclas vazias contendo pelo menos um caractere alocado a uma tecla adjacente.

A estrutura de vizinhança N_2 é um conjunto de soluções obtidas através da troca entre dois símbolos alocados. O conjunto de soluções que forma a vizinhança N_k é formado através de todas as permutações possíveis entre k símbolos alocados.

Os autores ainda adaptaram algumas técnicas usadas no QAP para diminuir a complexidade no cálculo do valor da função objetivo de uma nova solução obtida após ação das vizinhanças. Dessa forma, o procedimento se torna menos custoso computacionalmente.

As técnicas de aceleração desenvolvidas foram denominadas soluções equivalentes e *hashing*. Uma solução é dita canônica se há um caractere na linha mais acima e um na coluna mais à esquerda do arranjo e se o menor caractere em ordedm alfabética se encontra na tecla da posição mais acima e à esquerda do teclado. Uma solução equivalente, então, configura aquela obtida através do espelhamento vertical, rotação de 90º na direção anti-horária e translação. As soluções equivalentes têm o mesmo valor de função objetivo de uma solução canônica.

A técnica de *hashing* desenvolvida pelos autores consiste em armazenar um valor inteiro correspondente a cada solução diferente obtida. Para calcular esse valor, somam-se os valores das frequências de certos caracteres com cada caractere adjacente a eles. Os caracteres considerados encontram-se nas fronteiras da solução em posições específicas. Essas posições são as teclas mais à direita da fronteira superior, inferior na fronteira direita, mais à esquerda na fronteira inferior e superior na fronteira esquerda.

Para resolver esse problema, os autores adaptaram algumas meta-heurísticas usadas para o QAP: SA, TS, VNS, *fast ant system* (FANT). Um algoritmo de busca local chamado *LS-Refine* também foi desenvolvido, com o objetivo de refinar as soluções obtidas através dos outros métodos, levando-as a um ótimo local. Esse algoritmo funciona através da exploração da vizinhança N_1 até atingir um ótimo local para, em seguida, explorar-se a vizinhança N_2 a fim de se obter um novo ótimo local. Os melhores *layouts* alcançados para cada língua pesquisada estão ilustrados na Figura 12.

Bi, Smith e Zhai (2010) utilizam o algoritmo Metropolis, como no trabalho de Zhai,

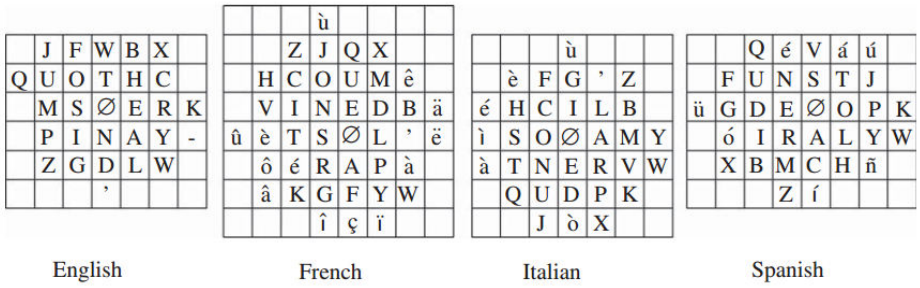


Figura 12: Teclados otimizados para inglês, francês, italiano e espanhol
Fonte: Dell'Amico et al. (2009), p. 10

Hunter e Smith (2002), utilizando a mesma equação *Fitts-digraph energy* para avaliar a eficiência de movimentação, porém com novos valores para os parâmetros a e b (0,083 e 0,127, respectivamente). Com o objetivo de facilitar as experiências de usuários do QWERTY com novos *layouts*, os autores criaram uma restrição de determina que caracteres não podem se mover mais do que uma tecla de distância de suas posições originais no *layout* QWERTY. Um experimento com 12 voluntários foi realizado, utilizando três diferentes tipos de *layouts*: QWERTY, um teclado arranjado conforme o método Metropolis sem a restrição de QWERTY, chamado de "livremente otimizado", e um teclado com a restrição mencionada, chamado de Quasi-QWERTY (Figura 13. Para o experimento, foram usadas 19 palavras em inglês, ordenadas aleatoriamente, que contêm em sua totalidade todas as letras do alfabeto.

q	w	d	r	t	u	y	l	k	p
z	a	s	e	h	n	i	o	m	
	x	f	v	c	g	b	j		

Figura 13: Teclado Quasi-QWERTY
Fonte: Bi, Smith e Zhai (2010), p. 2

Numa pesquisa posterior, Bi, Smith e Zhai (2012) adotaram o conceito de um *layout* definido considerando várias línguas ao mesmo tempo. Inicialmente, os autores desenvolveram um teclado combinando os idiomas inglês e francês. Posteriormente, um *layout* denominado K5 foi gerado, agrupando as particularidades das línguas inglesa, francesa, espanhola, alemã e chinesa. A metodologia utilizada pelos autores iniciou-se com a medição da média de tempo levado para digitar um caractere em várias línguas, minimizando-o através do algoritmo Metropolis (ZHAI; HUNTER; SMITH, 2002). O novo *layout* foi avaliado através do modelo *Fitts-digraph energy* (ZHAI; HUNTER; SMITH, 2002) com os parâmetros adotados no trabalho anterior dos mesmos autores (BI; SMITH; ZHAI, 2010). No escopo

de caracteres a serem alocados foram levadas em conta apenas as 26 letras do alfabeto ocidental, desconsiderando caracteres auxiliares, como o espaço. Para cada uma das propostas de combinação de idiomas dos autores, foi gerado um *layout* otimizado para todas os idiomas e um para cada idioma específico da combinação, de modo a comparar resultados teóricos. Os autores também consideraram a presença de caracteres diacríticos, que não existem na língua inglesa. Dois novos *layouts* foram criados com base no K5: um contendo todos os caracteres diacríticos, e um contendo teclas mortas com os acentos ortográficos, para modificarem as letras. A abordagem que contem os caracteres diacríticos gerou arranjos que permitem mais rápida digitação. Um teste nos mesmos moldes do trabalho de Bi, Smith e Zhai (2010) foi realizado para comparar o QWERTY e o K5, baseando-se nas experiências de usuários.

O trabalho de Dunlop e Levine (2012) tem como objetivo desenvolver um teclado que promovesse maior velocidade de digitação, menos erros e maior facilidade de aprendizagem para usuários do QWERTY. Os autores definiram uma frequência ponderada para todas os pares possíveis de caracteres, considerando 27 símbolos (26 caracteres da língua inglesa mais o espaço). Para resolver esse problema, os autores optaram por usar a fronteira de Pareto com os critérios de diminuir a distância de movimento de um dedo para aumentar a velocidade ao digitar, melhorar a clareza na interpretação de soletração, reduzindo os erros e usar uma versão flexibilizada da restrição de QWERTY do trabalho de Bi, Smith e Zhai (2010), ao penalizar teclas que se movimentem para posições muito distantes da original do QWERTY, mas permitindo deslocamentos maiores que uma tecla de distância. Esse último critério facilitaria a experiência de usuários do QWERTY. A lei de Fitts (Equação (1)) foi usada para avaliar a velocidade de digitação. O procedimento inicia-se com um conjunto de pontos gerados aleatoriamente e que passam por um processo de otimização local para diferentes combinações de pesos para os três critérios abordados. A cada iteração, o novo teclado gerado passa a fazer parte do conjunto se for melhor em algum critério ou se for tão bom quanto uma solução existente em todos os critérios. A fronteira de Pareto gerada contém 24000 teclados. Os autores também realizaram um teste com 10 usuários de dispositivos com *touchscreen* para avaliar a experiência dos usuários. Dois arranjos de caracteres foram produzidos pela metodologia dos autores. O primeiro deles se configura numa disposição trapezoidal (*Sath-trapezoidal*), ao passo que o segundo se trata de uma configuração retangular (*Sath-rectangular*), com teclas de tamanho maior. A Figura 14 ilustra o teclado Sath.

O trabalho de Alswaidan, Hosny e Najjar (2015) trata do desenvolvimento de um teclado para um dedo otimizado para o idioma árabe. O teclado é disposto em uma matriz 3 x 11, utilizando GA. Para atingir o objetivo do trabalho, foi necessário reunir

Figura 14: Teclado *Sath-trapezoidal*
Fonte: Dunlop e Levine (2012), p. 6

textos em árabe e determinar as frequências de cada par de caracteres. A função objetivo desenvolvida para o estudo consiste em três partes. A primeira parte é referente às frequências entre pares de caracteres, em que maior frequência leva a maior proximidade entre os dois símbolos. A segunda parte da função objetivo determina que letras mais frequentes tenderão a ser colocadas na fileira do meio. Essa linha de teclas possui peso 1, de acordo com a metodologia dos autores e, as fileiras superior e inferior têm pesos de 2 e 3, respectivamente. Esses pesos servirão de multiplicadores para as frequências dos pares de caracteres alocados. Como o idioma árabe é escrito da direita para a esquerda, os autores levaram em conta a direção de toque nas teclas na última parte da função objetivo, penalizando sequências de letras que forem posicionadas da esquerda para a direita, proporcionalmente às suas frequências. Um algoritmo genético (GA) foi desenvolvido , utilizando dois operadores *crossover* e três operadores de mutação. O primeiro operador *crossover* (TPX) escolhe dois pontos aleatórios dos pais e, para gerar o primeiro filho, as partes do pai_1 externas aos pontos de corte e as partes internas do pai_2 são unidas. Para gerar o segundo filho, os papéis dos pais se invertem. O segundo operador *crossover* (MUX) funciona com a criação de uma máscara, do tamanho do teclado, que determina, aleatoriamente, de que pai virá o caractere associado a certa tecla. O primeiro operador de mutação criado no trabalho de Alswaidan, Hosny e Najjar (2014) foi chamado *swap* e trata de trocar dois caracteres selecionados aleatoriamente. O segundo é chamado *insertion* e funciona selecionando uma tecla, aleatoriamente, inserindo o caractere da mesma em outra posição na mesma linha em que foi alocado, deslocando o restante dos caracteres. O terceiro operador de mutação foi um procedimento SA com as vizinhanças *swap* e *insertion* apresentadas. Os parâmetros usados no GA foram 50 para o tamanho da população, 0,7 para a taxa de *crossover*, 0,2 para taxa de mutação, 0,2 para taxa de substituição e, como critério de parada, foram definidas 20 iterações sem melhora. Os parâmetros do SA foram 1000 para temperatura inicial, 10 iterações por temperatura e taxa de resfriamento

de 0,95. Como a função objetivo desse trabalho é constituída de 3 diferentes critérios, os autores atribuíram pesos a cada um deles, sendo a configuração escolhida 0,7; 0,1 e 0,2 para distância, linha e direção de toque, respectivamente. Os melhores teclados gerados foram através das combinações de operadores TPX e *swap* e de MUX e *swap*.

Murali e Panicker (2016) também fizeram uso de GA e do algoritmo de decisão TOPSIS (*Technique for Order of Preference by Similarity to Ideal Solution*) para desenvolverem um teclado para o idioma inglês. Atributos de distância de fluxo (somatório dos produtos entre fluxo e distância Euclidiana de dois caracteres), média de palavras por minuto e porcentagem de aprendizagem foram considerados. O GA foi desenvolvido com um operador *crossover* de um ponto e um operador de mutação *swap* que troca dois caracteres de posição. Para selecionar em que cromossomos será aplicado *crossover* ou mutação, os autores consideraram o método da roleta. Os parâmetros usados no GA foram tamanho da população de 54, probabilidade de *crossover* de 0,8 e probabilidade de mutação de 0,1. A metodologia dos autores consiste em executar o algoritmo múltiplas vezes, separando as 10 melhores soluções. Os teclados são, então avaliados conforme os atributos previamente apresentados, determinando-se o melhor a cada execução. A ferramenta TOPSIS é usada para classificar as alternativas.

A Tabela 1 resume, cronologicamente, os trabalhos relacionados considerando teclados para um dedo, sendo dividida em colunas denominando os autores, ano de publicação e as principais abordagens usadas nos estudos desenvolvidos.

Tabela 1: Trabalhos relacionados que abordam *layouts* para teclados para um dedo

Autores	**Ano**	**Abordagem**
Mackenzie e Zhang	1999	Método Iterativo
Zhai, Hunter e Smith	2002	Simulação; Algoritmo Metropolis
Raynal e Vigouroux	2005	GA
Li, Chen e Goonetilleke	2006	Algoritmo Guloso + SA
Dell'Amico *et al.*	2009	LS; SA; TS; VNS; FANT
Bi, Smith e Zhai	2010	Algoritmo Metropolis
Bi, Smith e Zhai	2012	Algoritmo Metropolis
Dunlop e Levine	2012	Fronteira de Pareto
Alswaidan, Hosny e Najjar	2014	GA
Murali e Panicker	2016	GA + TOPSIS

2.2.2.2 Teclados para n dedos

Desenvolver teclados para n dedos possuem um histórico mais longo do que para um dedo, visto que seu conceito se iniciou com a criação de *layouts* para máquinas de escrever. Como esse não é o foco principal desse estudo, deu-se preferência aos trabalhos

de maior relevância para a área de desenvolvimento de *layouts* de teclado e que utilizassem abordagem ligada à Pesquisa Operacional.

Eggers et al. (2003) criaram o *keyboard arrangement problem* (KAP) e se basearam no trabalho de Marsan (1976) para desenvolver seis critérios heurísticos, de natureza ergonômica, para a avaliação de uma configuração de teclado para n dedos. O primeiro critério é denominado acessibilidade e carga e mede a variação de carga de digitação distribuída entre os dedos das mãos, considerando uma distribuição ideal. O segundo critério trata do número de teclas, avaliando a quantidade de teclas pressionadas para produzir um texto, visando minimizá-la. O terceiro critério aborda a alternância de mãos, determinando se há subutilização de alguma delas, visto que o mais rápido e confortável é alterná-las sempre que possível. O quarto critério avalia o uso consecutivo do mesmo dedo, calculando a quantidade de dígrafos acessados por um mesmo dedo de uma mão durante a produção de um texto. O quinto critério é definido com o objetivo de evitar grandes passos e consiste em um valor que considera os dígrafos digitados pela mesma mão (mas não mesmo dedo) que possuem distância vertical igual ou maior que 1 linha de teclas. O último critério trata da direção de toque e avalia a quantidade de dígrafos digitados pela mesma mão mas que não seguem a direção do dedo mindinho para o polegar. Os autores desenvolveram também um algoritmo ACO para resolução do KAP, gerando *layouts* para os idiomas inglês, francês e alemão.

Yin e Su (2011) abordaram o problema *general keyboard arrangement problem* (GKAP) que considera uma função que aborde arranjos de teclados destinados a diversas configurações, como teclados de único caractere, onde apenas um caractere é alocado a uma tecla; teclados de múltiplos caracteres, com mais de um caractere por tecla; teclados destinados ao uso de um dedo e para uso de n dedos. Os autores apresentaram uma função com quatro objetivos, baseando-se nos critérios de Eggers et al. (2003) para abordar aspectos ergonômicos de acessibilidade a teclas e conforto postural. O restante dos objetivos abordam eficiência da predição e desambiguação, tratando do acesso a teclas e choque de palavras. Todos os critérios foram adaptados para levar em conta os cenários de diferentes usos do teclado propostos pelos autores. Uma soma ponderada desses objetivos foi calculada, normalizando-os através da divisão por valores equivalentes relativos a um teclado de referência. Essa função é usada como função de aptidão para o algoritmo *cyber swarm* desenvolvido. Os autores utilizam também um a técnica de limitação (*bounding technique*) para o processo de avaliação de aptidão, reduzindo custo computacional. Esse algoritmo se diferencia do *particle swarm optimization* (PSO), no qual é baseado, pois a partícula (solução) é capaz de interagir com outros membros do conjunto de referência, determinando a melhor vizinhança a cada movimento. O algoritmo *cyber swarm* desenvol-

vido pelos autores conta com duas estratégias de diversificação. A primeira delas denota que quaisquer dois membros do conjunto de referência precisa de um limite mínimo de separação. A segunda estratégia de diversificação usa a técnica de *path relinking* para ligar regiões pouco exploradas do espaço de busca à melhor solução encontrada. Nesse trabalho, também foi desenvolvido um algoritmo de predição para reduzir a ambiguidade de palavras em teclados de múltiplos caracteres, bem como para reduzir o número total de acessos a teclas no geral. Os autores realizaram estudo empírico para determinar a interação homem-máquina. O experimento foi realizado com seis digitadores experientes.

Oulasvirta et al. (2013) propuseram um teclado em duas partes, para ser usado com os dois polegares, objetivando minimizar a distância percorrida e maximizar a alternância entre eles. Inicialmente, foi realizado estudo para determinar a melhor maneira de segurar um *tablet* em modo paisagem, sendo testadas 6 maneiras. Nessa fase inicial, também determinou-se o alcance de movimentação de um polegar. Essas avaliações serviram para determinar particularidades de projeto como o tamanho das teclas e das malhas de cada metade do teclado. Para determinar a performance de digitação com dois polegares, os autores usaram uma variação da lei de Fitts (Equação (1)) para duas fontes de toque, bem como testes com humanos. Os autores procuraram incorporar as seguintes conclusões obtidas nos testes realizados: os polegras precisam se alternar, sempre que possível, a mão não dominante deve ser favorecida em casos que precisem usar o mesmo polegar e, o lado da mão dominante precisa ter os caracteres mais frequentes alocados no centro, sempre que possível. Para o processo de otimização do teclado foi utilizado um método híbrido composto por fases de *gradient descent* e SA. A fase de *gradient descent* é executada para 5000 locais iniciais, sendo selecionadas as 100 melhores permutações. A fase de SA é executada a seguir por 10 vezes para cada solução candidata. *Gradient descent* é então executado novamente para os 10 melhores arranjos. O *layout* passa também por um processo de refinamento, no qual as linhas são movimentadas na direção horizontal, ocasionando melhoria muito pequena em palavras pos minuto. Os autores também desenvolvem um método de correção de erros e uma série de 6 testes para avaliar empiricamente o novo teclado denominado KALQ (Figura 15).

Karrenbauer e Oulasvirta (2014) desenvolveram uma formulação em programação inteira (PI) para o problema de alocação de letras a teclas. A função objetivo visa minimizar o custo de selecionar uma sequência de 2 caracteres, ponderado por uma probabilidade. Esse custo pode ser escrito como o produto entre o tempo de movimento entre duas teclas por variáveis binárias de decisão, que determinam se certos caracteres se localizam em determinadas teclas. As restrições incorporadas ao modelo determinam que um caractere só pode ser alocado a uma tecla e uma tecla, por sua vez, só pode receber um caractere, no

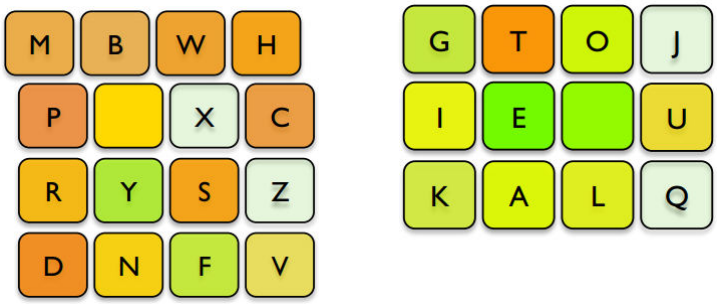


Figura 15: Teclado KALQ
Fonte: Oulasvirta et al. (2013), p. 6

máximo. Os autores apresentaram também duas técnicas de linearização: *reformulation-linearization*, que substitui os termos não lineares por novas variáveis de 4 índices, e a linearização de Kaufman-Broeckx, que usa variáveis de 2 índices que recebem valor 0 se a variável original associada é zero e contribuem com a função objetivo como uma função linear se a variável original for 1. Um algoritmo *branch-and-bound* foi utilizado para resolver três casos de *layouts* diferentes: o primeiro caso utilizou o arranjo de Dunlop e Levine (2012), o segundo usou a diagramação hexagonal de Zhai, Hunter e Smith (2002), ambos destinados a uso com um dedo e, por fim, o terceiro caso foi um teclado físico para n dedos. Para avaliar os casos de um dedo, usou-se a equação *Fitts-digraph energy*. Os melhores teclados gerados nesse trabalho estão ilustrados na Figura 16.

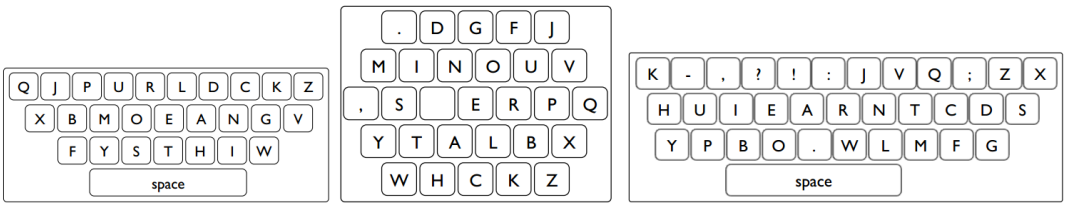


Figura 16: Casos 1, 2 e 3 gerados através de formulação de PI
Fonte: Karrenbauer e Oulasvirta (2014), p. 6

No trabalho de Işeri e Ekşioğlu (2015) foi desenvolvido um experimento com sete participantes visando estimar taxas de digitação de dígrafos em um teclado para n dedos de até 32 teclas e, a partir delas, determinar efeitos de dígrafo, coluna, linha, mão e período, além de custos de dígrafos. Nesse caso, o período se tratou de intervalos de 30 segundos de digitação. O experimento desenvolvido também almejou estabelecer relação da digitação de dígrafos com a fadiga muscular. O experimento consistiu na digitação de 241 dígrafos em um teclado físico enquanto os participantes eram avaliados quanto aos esforços musculares por meio de exame de eletromiograma. Para estimar as taxas de digitação de dígrafos, os autores fizeram uso de estatística descritiva e, para determinar os efeitos de dígrafo, coluna, linha, mão e período, foi usada análise de variância. Para determinar os custos de dígrafos, ou seja, o tempo levado para digitar uma letra e outra do

par, os autores fizeram cálculos estimativos levando em consideração os dados levantados no experimento, bem como dados apresentados em outros trabalhos relacionados. A segunda parte do trabalho de Işeri e Ekşioğlu (2015) foi otimizar um teclado para a língua turca, com base nos dados levantados experimentalmente. O problema foi abordado através de modelagem embasada no QAP e resolvido com a utilização de um algoritmo GA híbrido desenvolvido em trabalho relacionado. O melhor teclado gerado foi denominado *Turkish I-layout*.

A Tabela 2 resume, cronologicamente, os trabalhos relacionados considerando teclados para n dedos, sendo dividida em colunas denominando os autores, ano de publicação e as principais abordagens usadas nos estudos desenvolvidos.

Tabela 2: Trabalhos relacionados que abordam *layouts* para teclados para n dedos

Autores	**Ano**	**Abordagem**
Eggers *et al.*	2003	ACO
Yin e Su	2011	*Cyber Swarm*
Oulasvirta *et al.*	2013	*Gradient descent* + SA
Karrenbauer e Oulasvirta	2014	*Branch-and-Bound*
Işeri e Ekşioğlu	2015	GA híbrido

Capítulo 3

Procedimentos Metodológicos

Este capítulo apresenta a heurística proposta para resolução do SK-QAP e as instâncias desenvolvidas para a língua portuguesa. A primeira seção descreve o funcionamento do algoritmo, o qual é baseado no procedimento ILS. As seções subsequentes detalham algumas de suas características, como a representação da solução, o procedimento construtivo adotado, o método de busca local e as estruturas de vizinhança utilizadas e os mecanismos de perturbação incorporados. A última seção deste capítulo trata das instâncias da língua portuguesa e seu processo de desenvolvimento.

3.1 Algoritmo proposto

O algoritmo proposto é baseado no método ILS e denominado ILS-SKQAP, cujo funcionamento está descrito no Algoritmo 8. Os parâmetros recebidos pelo algoritmos são o número máximo de *restarts* (I_{MS}) e o número máximo de iterações do procedimento ILS (I_{ILS}). O ILS-SKQAP tem como primeiros passos a inicialização da função objetivo e da solução (linhas 2 e 3). Em seguida, inicia-se o laço de *restarts* (linha 4), em que, a cada iteração $Iter_{MS}$, uma solução inicial é gerada (linha 5) e atribuída à solução atual (linha 6) e a iteração atual do procedimento ILS ($Iter_{ILS}$) recebe valor 0 (linha 7). Os próximos passos (linhas 8-15) denotam a fase ILS do algoritmo, na qual o método RVND é aplicado à solução atual (linha 9) e, se houver melhora no valor da função objetivo, seu novo valor e a solução melhorada são armazenados e $Iter_{ILS}$ recebe o valor 0 (linhas 10-13). O ILS continua com a aplicação de perturbação à melhor solução encontrada no *restart* correntes e incremento de 1 a $Iter_{ILS}$ (linhas 14 e 15). Se, ao final do ILS houver melhora no valor da função objetivo, este é armazenado, bem como a solução associada a ele (linhas 16-18). A melhor solução é, então, retornada pelo algoritmo (linha 19).

Algoritmo 8 ILS-SKQAP

```
1: Procedimento ILS-SKQAP(I_MS, I_ILS)
2:     f* ← ∞
3:     s* ← ∅
4:     para Iter_MS = 1, ..., I_MS faça
5:         s ← GerarSoluçãoInicial()
6:         s' ← s
7:         Iter_ILS ← 0
8:         enquanto Iter_ILS ≤ I_ILS faça
9:             s ← RVND(s)
10:            se f(s) < f(s') então
11:                s' ← s
12:                f(s') ← f(s)
13:                Iter_ILS ← 0
14:            s ← Perturbação(s')
15:            Iter_ILS ← Iter_ILS + 1
16:        se f(s') < f* então
17:            s* ← s'
18:            f* ← f(s')
19:    retorne s*
20: fim Procedimento
```

3.2 Representação da solução

A solução gerada para o SK-QAP é um conjunto de caracteres, de tamanho definido pela instância abordada, distribuídos em uma malha retangular de teclas. Todas as teclas são quadradas e possuem as mesmas dimensões. A cada tecla será atribuído, no máximo, um caractere e cada caractere só poderá ocupar uma tecla da malha disponível.

O tamanho das instâncias, n, varia entre 27 caracteres, para as 5 instâncias aleatórias do idioma inglês (English_Rand_1 a 5) e 41 caracteres, para a instância French. As dimensões da malha foram definidas como 10 linhas e 14 colunas. Com essas dimensões garantiu-se espaço disponível o suficiente para acomodar os todos caracteres nos casos de cada instância, com a possibilidade de eliminação as teclas vazias.

Um *array* de duas dimensões $p \times q$ foi utilizado para representar a malha de teclas, com $p = 0, ..., 9$ e $q = 0, ...13$. Os caracteres da lista de cada idioma foram enumerados a partir de 0 até $n-1$. Como o número de espaços disponíveis é maior do que o tamanho das instâncias resolvidas, há ocorrência de espaços sem alocação, os quais são preenchidos com um valor -1. A impressão na tela gerada pelo algoritmo substitui os valores numéricos das teclas com alocação pelos caracteres correspondentes e os valores de teclas vazias por um símbolo $*$. A Figura 17 apresenta um exemplo de representação de solução do SK-QAP e a Figura 18 apresenta uma versão renderizada desta solução.

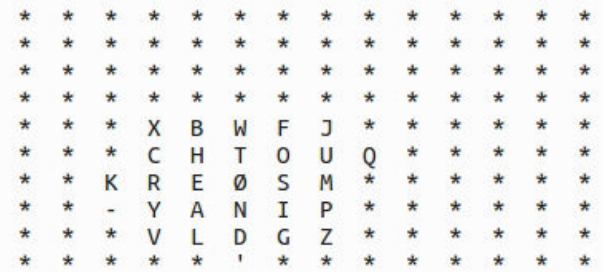


Figura 17: Solução do algoritmo proposto

	X	B	W	F	J	
	C	H	T	O	U	Q
K	R	E	Ø	S	M	
-	Y	A	N	I	P	
	V	L	D	G	Z	
			'			

Figura 18: Solução renderizada do algoritmo proposto

3.3 Procedimento construtivo

O procedimento construtivo desenvolvido nesse trabalho possui componentes aleatórios e gulosos. A aleatoriedade se faz necessária para assegurar mais detalhada exploração do espaço de busca ao promover diferentes estados iniciais e o componente guloso garante menor esforço nas fases iniciais da busca local posteriormente aplicada.

O Algoritmo 9 descreve o funcionamento do procedimento de construção da solução inicial. Os primeiros passos do procedimento são a inicialização da lista de caracteres L candidatos à alocação (linha 2), da matriz de teclas M, que deve estar vazia, a princípio (linha 3) e da solução s (linha 4). Em seguida, a tecla central da matriz M é determinada (linhas 5 e 6) e esta tecla recebe o caractere espaço que é, então, retirado de L (linhas 7 e 8). Esses passos foram incorporados ao algoritmo pela observação anterior de que o caractere espaço é o mais frequente em todas as instâncias, o que o torna um candidato propício para o centro do teclado e para, consequentemente, estar localizado a menores distâncias dos outros caracteres. Os próximos passos do algoritmo introduzem o componente aleatório do procedimento. Quatro caracteres (c_1 a c_4) são escolhidos, aleatoriamente, em L e alocados nas teclas imediatamente acima, abaixo, à direita e à esquerda da tecla central, sendo posteriormente excluídos da lista de candidatos (linhas 9 a 14). O custo preliminar da solução com essas 5 alocações é então calculado na linha 15. As linhas 16 a 26 descrevem o componente guloso do algoritmo, no qual os caracteres restantes de L vão sendo escolhidos, um a um, e testados nas teclas vazias de M, sendo guardado o melhor custo obtido dentre os testes de alocação e a tecla associada a este custo. Após todas as teclas terem sido

testadas, o caractere testado é alocado na posição referente ao menor custo obtido e retirado de L. Esses passos se repetem até que todos os caracteres tenham sido alocados e L esteja vazia. O procedimento termina com a atribuição de M à solução s (linha 27) que é retornada pelo algoritmo (linha 28).

Algoritmo 9 Gerar Solução Inicial

```
 1: Procedimento GerarSoluçãoInicial
 2:     Inicialize L;                                   ▷ Lista de caracteres candidatos
 3:     Inicialize M(p × q) com −1;                     ▷ Matriz de teclas
 4:     s ← ∅;
 5:     L_c ← (p/2);                                    ▷ Linha central
 6:     C_c ← (q/2);                                    ▷ Coluna central
 7:     M(L_c, C_c) ← espaco;
 8:     L ← L−{espaco};
 9:     Selecione 4 caracteres aleatoriamente (c_1, c_2, c_3, c_4);
10:     M(L_c, (C_c − 1)) ← c_1;
11:     M(L_c, (C_c + 1)) ← c_2;
12:     M((L_c − 1), C_c) ← c_3;
13:     M((L_c + 1), C_c) ← c_4;
14:     L ← L−{c_1, c_2, c_3, c_4};
15:     Calcule iniCost;                                ▷ Custo do estado inicial da solução
16:     enquanto L não está vazia faça
17:         MinCost ← ∞;
18:         para i = 0, ..., p faça
19:             para j = 0, ..., q faça
20:                 para c = 0, ..., |L − 1| faça
21:                     CurrCost ← iniCost+custo de inserção de c em M(i, j);
22:                     se CurrCost < MinCost então
23:                         Guarde os valores de c, i e j;
24:                         MinCost ← CurrCost;
25:         M(i, j) ← c;
26:         L ← L−{c};
27:     s ← M;
28:     retorne s;
29: fim Procedimento
```

3.4 Busca local

O algoritmo de busca local proposto utiliza o *randomized variable neighborhood descent* (RVND) como apresentado no trabalho de Subramanian et al. (2010). Esse método é uma variação do *variable neighborhood descent* (VND) proposto por Mladenović e Hansen (1997). O funcionamento do RVND se dá através de uma lista de estruturas de vizinhança, escolhendo uma delas, aleatoriamente. Se a nova solução obtida após ação dessa vizinhança trouxer melhoras ao valor da função objetivo, todas as estruturas da lista ficam disponíveis na próxima iteração. Caso contrário, a estrutura explorada é retirada da lista de vizinhanças acessíveis na iteração seguinte e nova estrutura é aleatoriamente escolhida. O procedimento termina se todas as estruturas de vizinhança foram exploradas

mas o valor da função objetivo não sofreu melhoras.

Nesse estudo, três estruturas de vizinhança foram utilizadas. Duas delas foram incorporadas a partir do trabalho de Dell'Amico et al. (2009): *contour filling* ($N^{(1)}$) e *pairwise-exchange* ($N^{(2)}$). $N^{(1)}$ funciona através do movimento de um caractere para uma posição de contorno. Nesse caso, uma posição de contorno é definida como uma tecla vazia em cuja adjacência há pelo menos uma tecla com caractere alocado. A vizinhança $N^{(2)}$ seleciona dois caracteres e realiza a troca de suas posições no teclado. O funcionamento dessas estruturas de vizinhança são ilustrados nas Figuras 19 e 20, respectivamente.

Figura 19: *Contour filling* ($N^{(1)}$)

Figura 20: *Pairwise-exchange* ($N^{(2)}$)

Para computar a variação do valor da função objetivo de maneira eficiente em movimentações na vizinhança $N^{(1)}$, foram usadas as Equações (6) e (7), como apresentadas por Dell'Amico et al. (2009). A Equação (6) define o cálculo da variação de valor da função objetivo ao mover-se um caractere i para uma posição de contorno j, gerando uma solução φ' partir de uma solução φ. Já a Equação (7), calcula a variação do valor

da função objetivo após movimentação de um caractere h para uma posição de contorno l, gerando a solução φ'' a partir de φ'. A condição para o uso da Equação (7) é que a posição de contorno h pertença ao conjunto de posições de contorno de φ e φ'. Caso se trate de uma posição de contorno que surge após a movimentação de i para j, a Equação (6) deve ser usada. Através da Equação (6), $z(\varphi')$ é calculada em $O(n)$ operações e, com a Equação (7), $z(\varphi'')$ é computada com complexidade $O(1)$ (DELL'AMICO et al., 2009). Para tirar vantagem dessas equações e reduzir a complexidade de cálculo da função objetivo em um movimento seguinte de $N^{(1)}$, esta vizinhança foi implementada com o método de primeiro vizinho com melhora, em que os movimentos são realizados com base na primeira melhora obtida.

$$\Delta(\varphi, i, j) = z(\varphi') - z(\varphi) = \sum_{\substack{k=1 \\ k \neq i}}^{n} f_{ki}(d_{\varphi_k j} - d_{\varphi_k \varphi_i}) \tag{6}$$

$$\Delta(\varphi', h, l) = z(\varphi'') - z(\varphi') = \Delta(\varphi, h, l) + f_{ih}(d_{jl} - d_{\varphi_i l} + d_{\varphi_i \varphi_h} - d_{\varphi_h j}) \tag{7}$$

A variação de valor da função objetivo após movimentos em $N^{(2)}$ foi eficientemente calculada usando a Equação (8), também apresentada no trabalho de Dell'Amico et al. (2009). Nesse caso, φ' é uma solução obtida a partir de φ após troca de posições entre um par de caracteres r e s. Com o uso da Equação (8), $z(\varphi')$ é computado em $O(n)$ operações.

$$\Delta(\varphi, r, s) = z(\varphi') - z(\varphi) = \sum_{\substack{k=1 \\ k \neq r,s}}^{n} (f_{rk} - f_{sk})(d_{\varphi_s \varphi_k} - d_{\varphi_r \varphi_k}) \tag{8}$$

A terceira estrutura de vizinhança utilizada foi desenvolvida neste trabalho e denominada *two pairs swap* ($N^{(3)}$). Seu funcionamento se dá por meio da seleção de dois pares de caracteres adjacentes e troca de suas posições no teclado. Os pares de caracteres podem ser adjacentes no sentido vertical ou horizontal e suas posições relativas são mantidas após a realização da troca. O cálculo eficiente da variação do valor da função objetivo, em movimentos de $N^{(3)}$, foi realizado através da adaptação de equação proposta por Frieze et al. (1989) para o QAP. No caso do SK-QAP, φ' é a solução obtida através da troca entre caracteres r e s e o valor de $z(\varphi')$ é calculado usando a Equação (8). A troca entre caracteres u e v gera uma solução φ'' e a Equação (9) é utilizada para calcular

$z(\varphi'')$ com complexidade $O(1)$.

$$\begin{aligned}\Delta(\varphi', u, v) &= z(\varphi'') - z(\varphi') \\ &= \Delta(\varphi, u, v) + (f_{ru} - f_{rv} + f_{sv} - f_{su})(d_{\varphi_s\varphi_u} - d_{\varphi_s\varphi_v} + d_{\varphi_r\varphi_v} - d_{\varphi_r\varphi_u}) \qquad (9)\end{aligned}$$

O funcionamento do *two pairs swap* é ilustrado nas Figuras 21 e 22, para os casos de pares adjacentes nos sentidos vertical e horizontal, respectivamente.

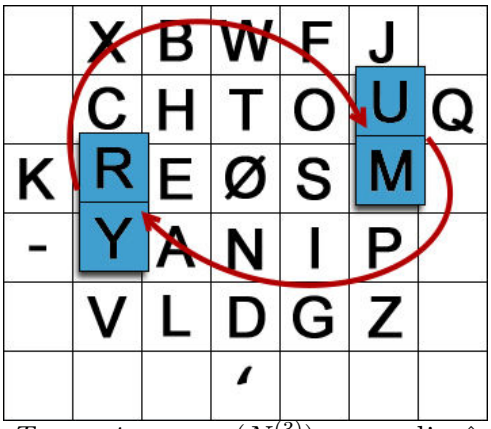


Figura 21: *Two pairs swap* ($N^{(3)}$) com adjacência vertical

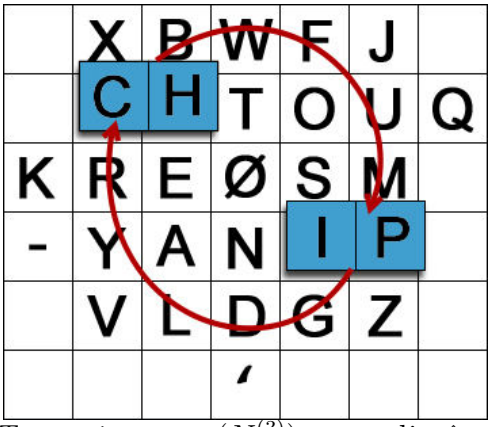


Figura 22: *Two pairs swap* ($N^{(3)}$) com adjacência horizontal

As estruturas de vizinhança $N^{(2)}$ e $N^{(3)}$ utilizam o método do melhor vizinho, no qual se realiza apenas o movimento que gera a maior melhora à função objetivo. Nas três vizinhanças, o procedimento de atualização da estrutura da solução é feito em tempo constante.

3.5 Mecanismos de perturbação

Mecanismos de perturbação foram desenvolvidos para contribuir com o processo de diversificação do algoritmo proposto, escapando de ótimos locais. Um bom método ILS precisa ter mecanismos de perturbação cuidadosamente selecionados de forma a atingir os resultados esperados de melhoras da solução na iteração subsequente. Se uma perturbação

for muito pequena, a solução pode retornar a um estágio de ótimo local anterior. Se, por outro lado, uma perturbação for muito grande, a nova solução obtida se torna uma solução gerada aleatoriamente, perdendo características promissoras da solução anterior (LOURENÇO; MARTIN; STÜTZLE, 2010; GUTIÉRREZ; BRIZUELA, 2006; SOUZA, 2011).

Os mecanismos de perturbação desenvolvidos neste trabalho foram denominados *ejection chain* ($P^{(1)}$) e *multiple pairwise-exchange* ($P^{(2)}$). A perturbação $P^{(1)}$ seleciona, aleatoriamente, um conjunto de n caracteres e troca suas posições entre eles, dessa forma: o primeiro caractere selecionado assume a tecla do próximo e assim sucessivamente, até o último caractere assumir a posição anteriormente ocupada pelo primeiro. O funcionamento desse mecanismo é ilustrado na Figura 23.

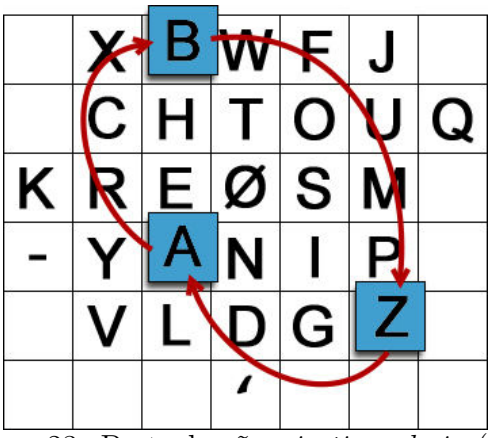


Figura 23: Perturbação *ejection chain* ($P^{(1)}$)

A perturbação $P^{(2)}$ seleciona n pares de caracteres e troca suas posições. O funcionamento desse mecanismo é ilustrado na Figura 24.

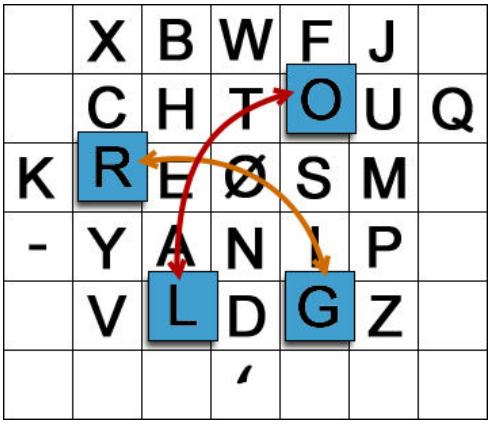


Figura 24: Perturbação *multiple pairwise-exchange* ($P^{(2)}$)

Para $P^{(1)}$ limitou-se o valor de n de 3 a 5 caracteres. Já para $P^{(2)}$, os valores limitantes de n foram de 2 a 3 pares.

3.6 Geração de instâncias da língua portuguesa

Os conjuntos de instâncias para inglês, espanhol, italiano e francês usados neste trabalho foram originalmente desenvolvidas no trabalho de Dell'Amico et al. (2009). O conjunto de instâncias referentes à língua portuguesa foi desenvolvida para este trabalho seguindo processo similar ao apresentado por Dell'Amico et al. (2009). Dessa forma, garantiu-se homogeneidade na grandeza dos valores das instâncias.

Para desenvolver a instância real do idioma português, foram listadas as 10000 palavras mais frequentes da língua, de acordo com o corpus brasileiro, disponibilizado pela PUC-SP (`http://corpusbrasileiro.pucsp.br/`). A partir dessa lista, a frequência de cada par de caracteres consecutivos foi computada. A ordem de caracteres em um par não é diferenciativa, ou seja, a frequência para um par "AE", por exemplo, é somada àquela de um par "EA". Para computar o caractere espaço, foi considerada a presença do mesmo antes e depois de cada palavra da lista. A instância real do idioma português foi denominada Portuguese, mantendo padronização proposta por Dell'Amico et al. (2009).

Um conjunto de 5 instâncias aleatórias também foi criada. Foram listadas as 5000 palavras mais frequentes de acordo com o corpus brasileiro e, desta lista, foram selecionadas, aleatoriamente, 1000 palavras e suas frequências. A partir da nova lista, realizou-se processo similiar à geração da instância real, computando os pares de caracteres encontrados e suas frequências. As instâncias aleatórias desenvolvidas para a língua portuguesa foram denominadas Portuguese_Rand_1 a 5, mantendo padronização de Dell'Amico et al. (2009).

Os caracteres acentuados presentes na lista de palavras foram computados como caracteres diferentes daqueles sem o diacrítico. A decisão foi tomada com base no formato encontrado nas instâncias existentes do problema.

A Tabela 3 mostra uma lista dos caracteres encontrados no conjunto de palavras selecionados do corpus brasileiro para instância real, juntamente com suas frequências correspondentes.

Tabela 3: Lista de caracteres da língua portuguesa e suas frequências

	Caractere	**Frequência**		**Caractere**	**Frequência**
1	ø	1422320358	21	Q	56479112
2	E	794640504	22	Ç	53842718
3	A	781153634	23	H	46140218
4	O	700213656	24	É	32280960
5	S	490654782	25	Á	25258100
6	R	414160132	26	Í	22205040
7	I	412819244	27	Z	20684168
8	D	397200672	28	J	19908382
9	N	335387212	29	X	16918044
10	T	318277318	30	Ê	12105690
11	M	275514250	31	Ó	11613906
12	C	230794514	32	Õ	10605416
13	U	230347514	33	Ú	8496554
14	P	195212434	34	À	7709438
15	L	182716622	35	W	4300930
16	V	79217796	36	-	3687032
17	F	75512502	37	Â	3676518
18	Ã	67358752	38	Y	3346778
19	G	65734404	39	K	2452552
20	B	58119094	40	Ô	2226930

Capítulo 4

Resultados Computacionais

Este capítulo apresenta os resultados obtidos pelo ILS-SKQAP e faz uma comparação com outros resultados reportados na literatura. O ILS-SKQAP foi implementado em C++ (g++ 5.4.0) e os experimentos executados em um Intel ®Core™ i7-3770 com 3,40 GHz e 16 GB de memória RAM, em sistema operacional Ubuntu 16.04. Testes de impacto realizados para a estrutura de vizinhança adaptada do QAP neste trabalho e para os mecanismos de perturbação adotados são descritos e seus resultados discutidos.

4.1 Impacto da vizinhança *Two Pairs Swap*

Para analisar o impacto da vizinhança *two pairs swap* foi realizado um teste com duas etapas. Durante a primeira etapa foram utilizadas as vizinhanças $N^{(1)}$ e $N^{(2)}$. Na segunda etapa do teste, a vizinhança $N^{(3)}$ foi adicionada às configurações anteriores. Dessa forma foi possível avaliar o funcionamento do algoritmo nas condições de ausência e presença da nova vizinhança desenvolvida.

Para esse teste, os valores dos parâmetros I_{MS} e I_{ILS} foram definidos como 1 e a fase de perturbações do algoritmo foi ignorada. Observou-se que, ao adicionar a estrutura de vizinhança $N^{(3)}$, há melhor desempenho do algoritmo em 75% das instâncias, ainda com tempos computacionais muito próximos àqueles obtidos nos testes sem essa estrutura, registrando até tempos menores em 33,3% das instâncias. Ao considerar-se classes de instâncias, agrupando-as por idiomas, é possível perceber que ocorrem melhoras nos *gaps* obtidos em todos os casos. A Figura 25 ilustra o *gap* médio obtido para cada classe de instâncias e a Figura 26 mostra os tempos computacionais médios obtidos em cada uma destas classes, considerando as duas etapas do teste de impacto.

Através de análise dos resultados, optou-se por incorporar a estrutura de vizinhança ao ILS-SKQAP, já que seu comportamento na obtenção de *gaps* foi bastante promissor.

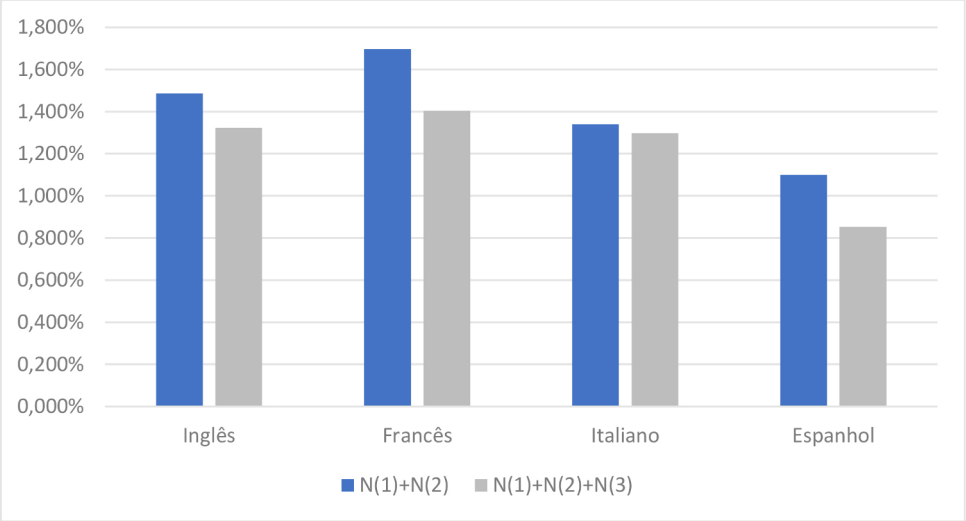


Figura 25: *Gaps* médios das soluções no teste de impacto de $N^{(3)}$

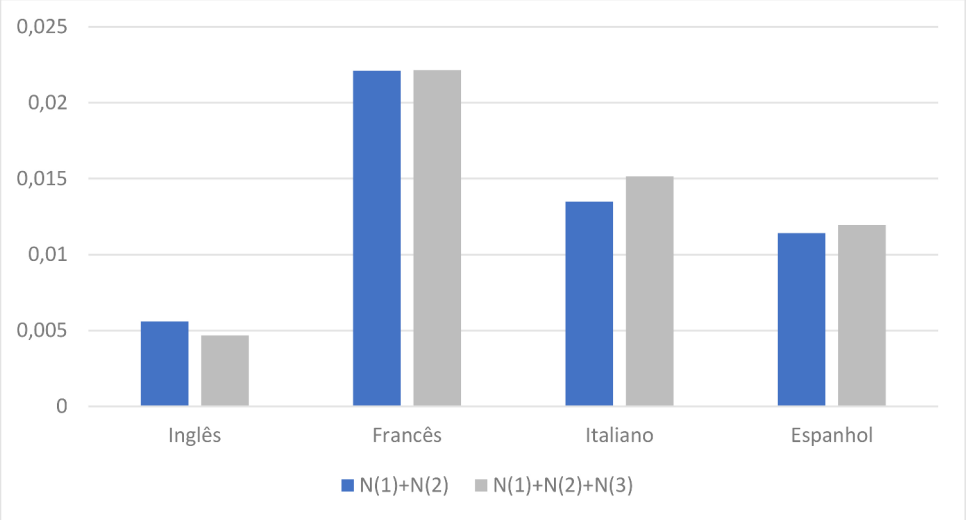


Figura 26: Tempos médios de CPU (s) no teste de impacto de $N^{(3)}$

4.2 Impacto dos mecanismos de perturbação

Para determinar o impacto dos mecanismos de perturbação desenvolvidos para o ILS-SKQAP, foi conduzido um teste composto por três etapas. A primeira etapa utilizou somente a perturbação *ejection chain* ($P^{(1)}$), a segunda etapa usou apenas a *multiple pairwise-exchange* ($P^{(2)}$), finalmente, a terceira etapa do teste ($P^{(1)} + P^{(2)}$) empregou ambos os mecanismos. Dessa maneira, foi possível avaliar o funcionamento do algoritmo nos casos dos mecanismos de perturbação individuais, bem como no caso em que há a combinação de ambos.

Para este teste o parâmetro I_{ILS} recebeu valor igual ao tamanho de cada instância (n) e I_{MS} recebeu valor 1. Percebeu que, em 50% das instâncias do SK-QAP, a utilização de ambos os mecanismos de perturbação oferece menores *gaps* médios entre as soluções

obtidas e as melhores soluções reportadas na literatura. As etapas contendo apenas as perturbações individuais resultaram em menores *gaps* para 25% das instâncias, em cada caso. A Figura 27 ilustra o *gap* médio obtido para cada classe de instâncias e a Figura 28 mostra os tempos computacionais médios obtidos em cada uma destas classes, considerando as duas etapas do teste de impacto das perturbações.

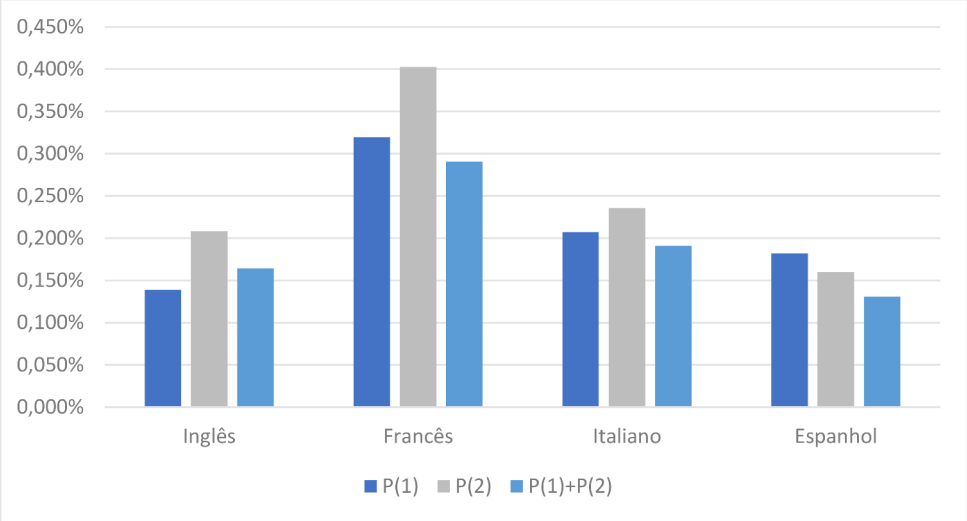


Figura 27: *Gaps* médios no teste de impacto de $P^{(1)}$ e $P^{(2)}$

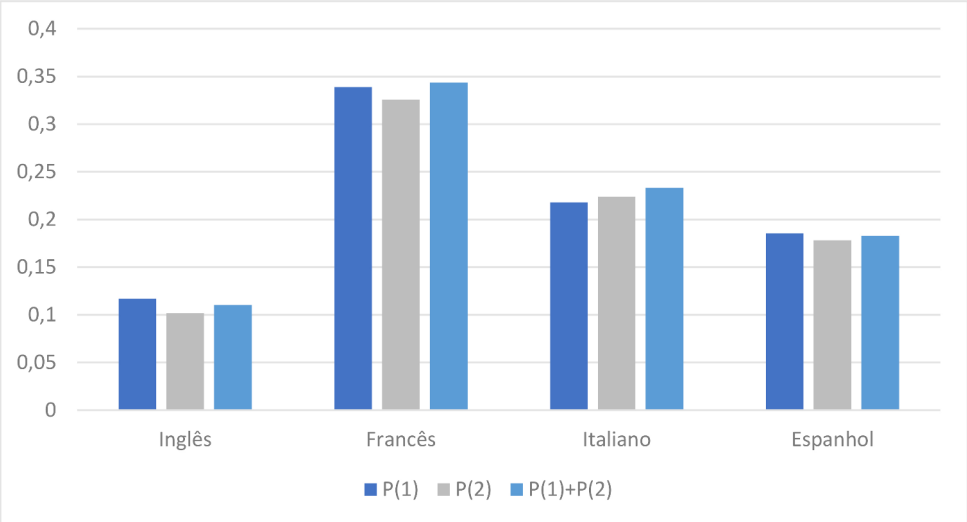


Figura 28: Tempos médios de CPU (s) no teste de impacto de $P^{(1)}$ e $P^{(2)}$

Mediante análise destes resultados, optou-se por incorporar ambos mecanismos de perturbação propostos ao ILS-SKQAP, visto que foi registrado menor *gap* médio com $P^{(1)} + P^{(2)}$ em três dos quatro conjuntos de instâncias disponíveis, em tempos computacionais competitivos, não ultrapassando 0,35 segundos.

4.3 Calibração de parâmetros

Para calibrar os parâmetros I_{MS} e I_{ILS} do ILS-SKQAP, foram conduzidos 6 testes, com diversos valores, a fim de estudar o comportamento do algoritmo, em termos de *gap* médio das soluções e tempo computacional demandado. Para cada configuração, o algoritmo foi executado 10 vezes. A Tabela 4 resume os valores assumidos por cada parâmetro em cada teste realizado. Os valores de I_{ILS} foram relacionados com o tamanho de cada instância (n).

Tabela 4: Valores dos parâmetros escolhidos para calibração.

I_{MS}	I_{ILS}
10	$4n$
10	$5n$
10	$6n$
15	$4n$
15	$5n$
15	$6n$

O procedimento de calibração de parâmetros utilizou apenas o conjunto de instâncias em francês, pois se trata do conjunto com instâncias de maior tamanho e, portanto, de maior dificuldade de resolução.

As Figuras 29 e 30 apresentam os *gaps* encontrados entre os custos das soluções e o tempo computacional médio para cada combinação de parâmetros testada, respectivamente.

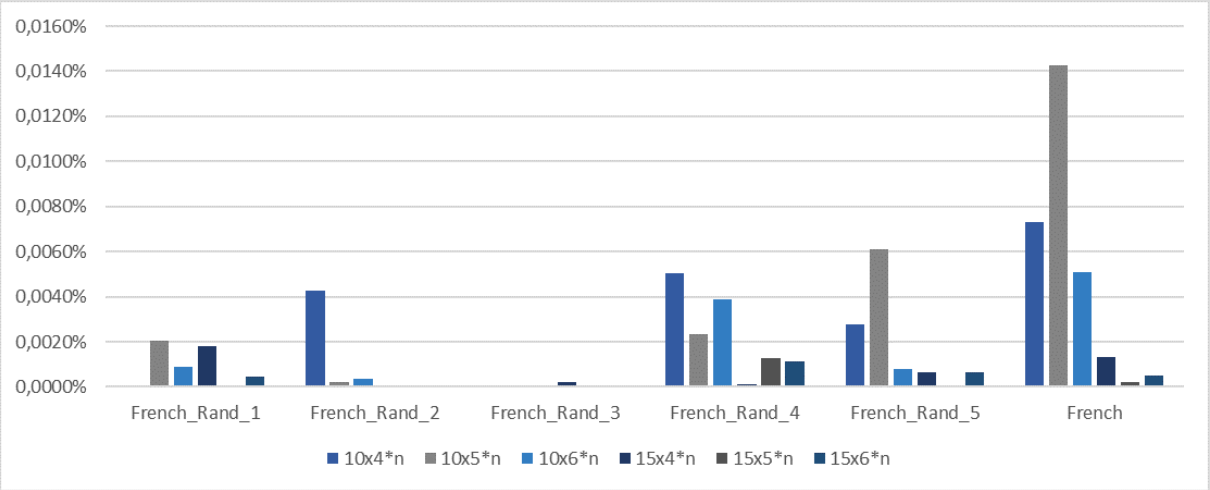


Figura 29: *Gaps* médios das soluções nos testes de calibração de parâmetros

Através desses resultados, definiu-se os valores de I_{MS} e I_{ILS} como 15 e $4n$, respectivamente. Essa combinação oferece menores diferenças de custos entre as soluções obtidas

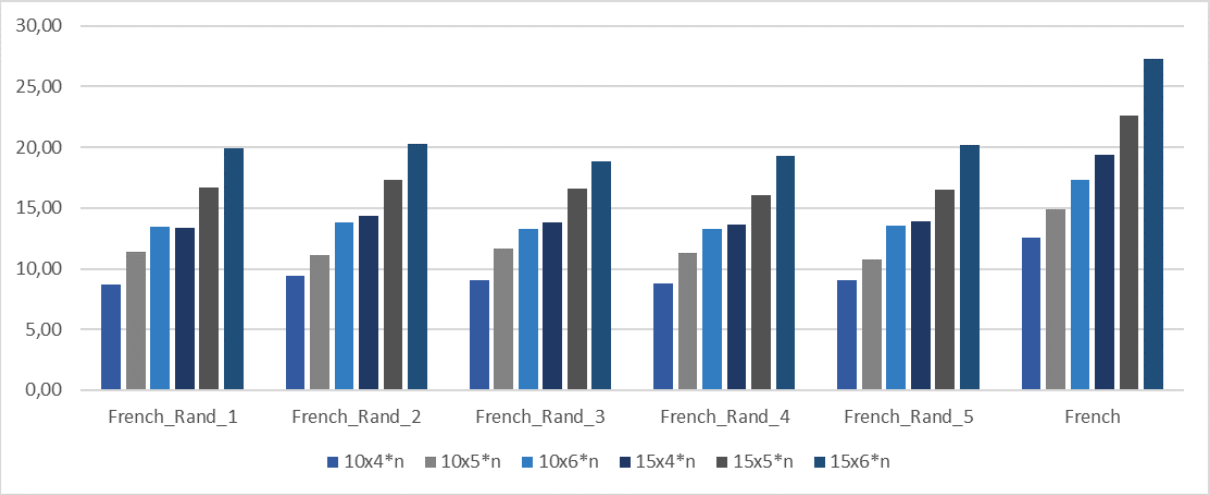


Figura 30: Tempos médios de CPU (s) nos testes de calibração de parâmetros

e as melhores soluções reportadas na literatura, sempre abaixo de 0,002%. Também proporciona tempos computacionais competitivos, com um pior caso de 19,36 segundos para a instância French.

4.4 Resultados para as instâncias existentes

As instâncias utilizadas nos experimentos computacionais do ILS-SKQAP foram propostas no trabalho de Dell'Amico et al. (2009) e resolvidas pelos autores através de algoritmos SA, TS, VNS e FANT. Os melhores resultados foram obtidos com o FANT, com parâmetro de reforço da trilha de feromônios R igual a 8, sendo capaz de encontrar as melhores soluções das 24 instâncias em um tempo de execução máximo de 120 segundos.

A Tabela 5 reúne os resultados obtidos pelo ILS-SKQAP, com a parametrização selecionada de $I_{MS} = 15$ e $I_{ILS} = 4n$, para as instâncias existentes dos idiomas inglês, francês, italiano e espanhol. As definições do rótulos das colunas da Tabela 5 estão listadas a seguir.

- **Instance** - 24 instâncias propostas por Dell'Amico et al. (2009), agrupadas em classes de instância, sendo a primeira baseada em dados reais e as 5 subsequentes geradas aleatoriamente.
- **BKS** - Valores associados às melhores soluções conhecidas para cada instância.
- **ILS-SKQAP** - Valores associados às melhores soluções encontradas pelo algoritmo ILS-SKQAP.

- **Avg. Sol.** - Média dos valores associados às melhores soluções encontradas em 10 execuções do ILS-SKQAP.
- **Avg. Time (s)** - Média dos tempos computacionais demandados em 10 execuções do ILS-SKQAP, em segundos.
- **Avg. Gap (%)** - Médias dos *gaps* obtidos entre os valores associados às melhores soluções encontradas pelo ILS-SKQAP e aqueles associados às melhores soluções conhecidas.
- **#Best** - Número de vezes em que o ILS-SKQAP encontrou a melhor solução conhecida, em 10 execuções.

O *site* CPU Benchmark (`https://www.cpubenchmark.net/`) foi usado para comparar melhor a eficiência do ILS-SKQAP com outros resultados para as mesmas instâncias, através da estimação de fator comparativo de CPU com base em valores de desempenho por uma única *thread*. Estimou-se que o Intel Pentium 4 3,0 GHz utilizado no trabalho de Dell'Amico et al. (2009) é 3,18 vezes mais lento que o Intel i7 3,40 GHz utilizado neste trabalho. Através desse fator, o tempo máximo de execução determinado por Dell'Amico et al. (2009) seria traduzido em de 37,73 segundos, que é um valor 94,9% maior que o pior tempo de 19,36 segundos alcançado pelas configurações adotadas para o ILS-SKQAP. Pode-se perceber, que o ILS-SKQAP tem desempenho superior, em termos de custo de CPU, alcançando média de 14,75 segundos para as instâncias do idioma francês, a classe mais difícil. O algoritmo proposto demonstrou robustez ao ser capaz de alcançar as melhores soluções conhecidas de todas as 24 instâncias, na maioria das execuções, obtendo *gaps* médios menores que 0,001% em todas as classes de instâncias, com destaque para *gaps* médios inferiores a 0,0007% quando agrupadas as instâncias do idioma francês.

Outra análise dos resultados foi feita através de geração de gráficos de convergência empírica (AIEX; RESENDE; RIBEIRO, 2007). Para gerá-los, o ILS-SKQAP foi executado 100 vezes, tendo como critério de parada a obtenção da melhor solução conhecida. As Figuras 31, 32, 33 e 34 ilustram os gráficos de convergência empírica do algoritmo para as instâncias English, French, Italian e Spanish, respectivamente. Pode-se observar que, para English, há alta probabilidade de encontrar a melhor solução em menos de 1 segundo. Para Italian e Spanish, em mais de 90% das execuções, o algoritmo encontrou a melhor solução em torno de 2 e 4 segundos, respectivamente. Para French, a instância mais difícil do SK-QAP, a melhor solução foi encontrada, em 80% das execuções, em torno de 1 minuto.

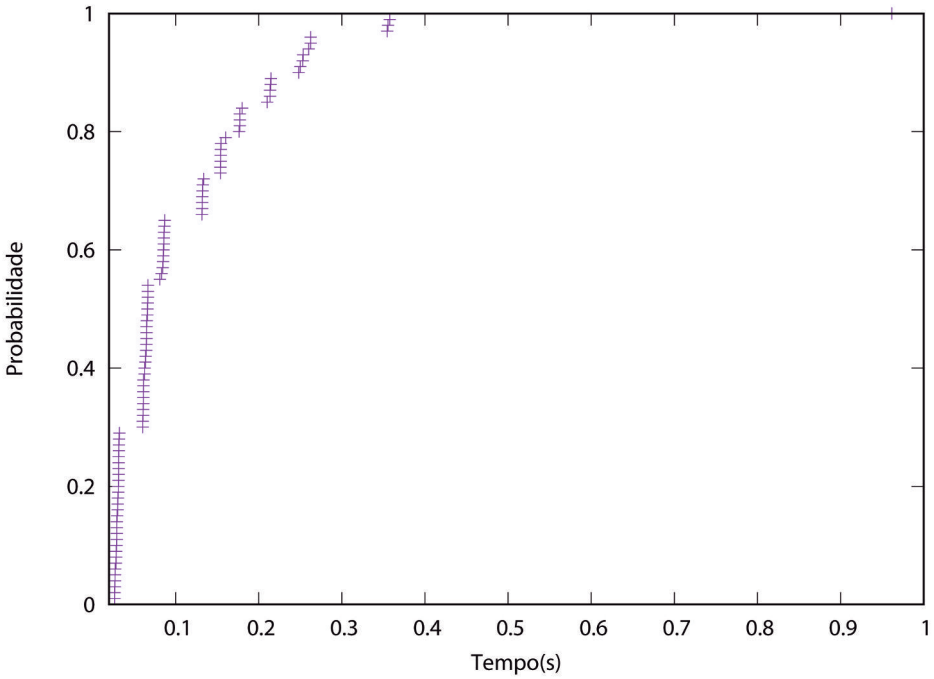


Figura 31: Gráfico de convergência para English

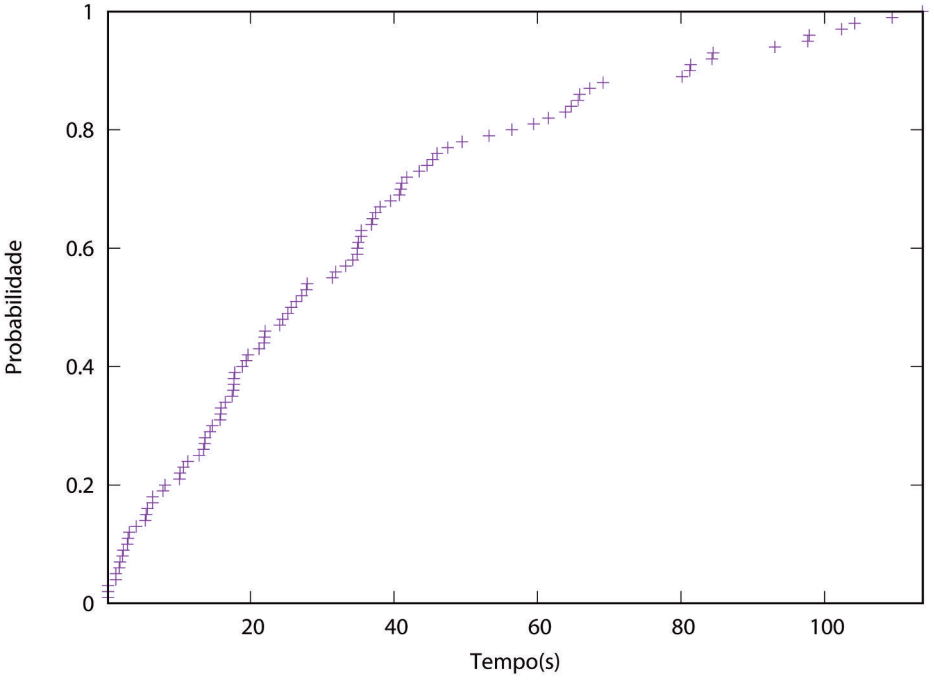


Figura 32: Gráfico de convergência para French

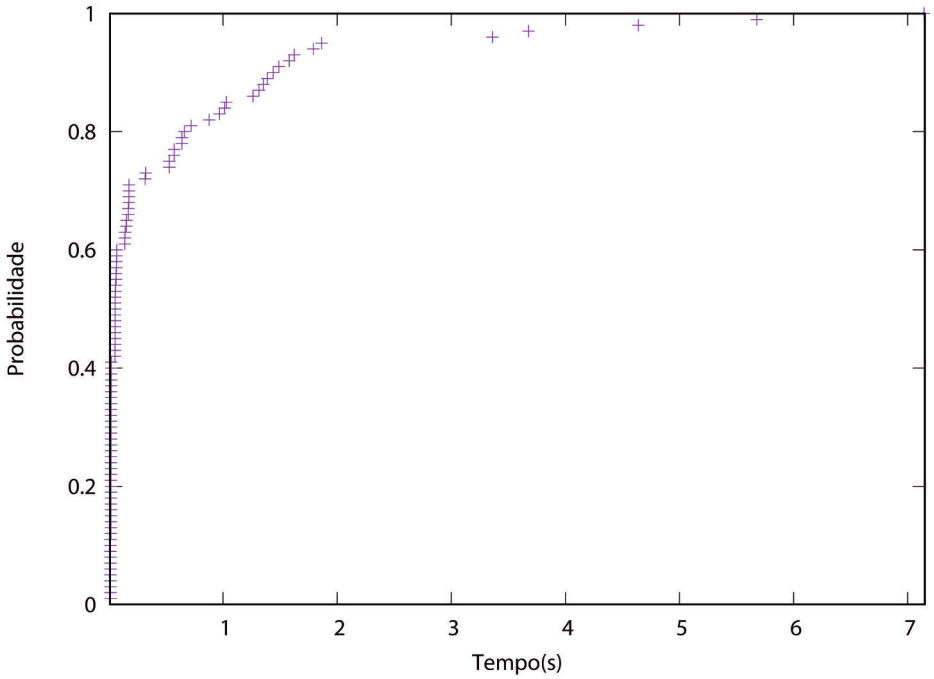


Figura 33: Gráfico de convergência para Italian

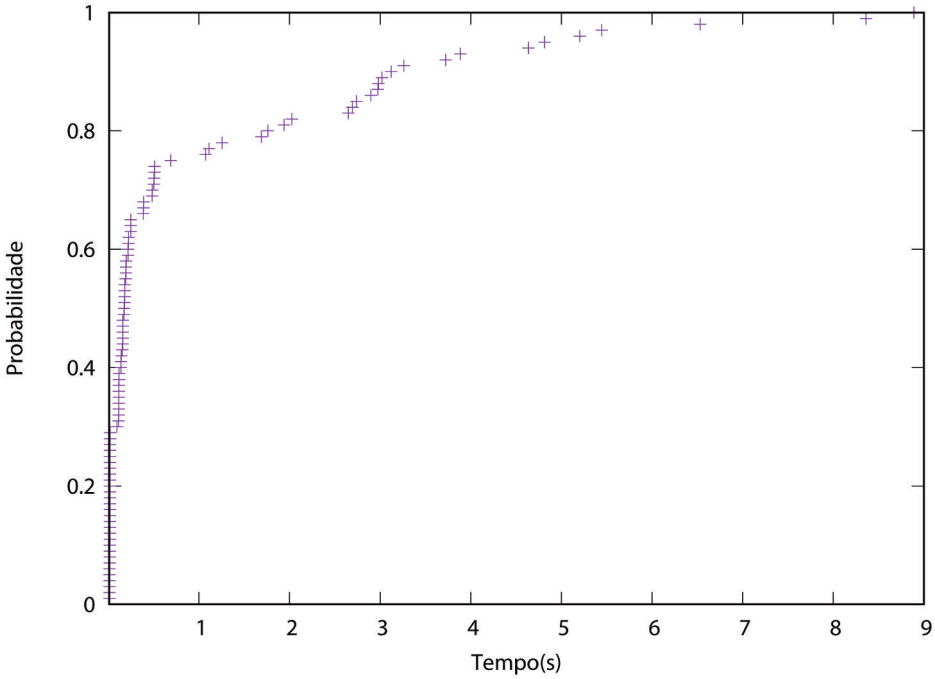


Figura 34: Gráfico de convergência para Spanish

Tabela 5: Resultados do ILS-SKQAP por instância existente

Instance	**BKS**	**ILS-SKQAP**	**Avg. Sol.**	**Avg. Time (s)**	**Avg. Gap (%)**	**#Best**
English	1199070166	1199070166	1199070166	4,074	0,000	10
English_Rand_1	647405083	647405083	647405083	2,413	0,000	10
English_Rand_2	655664577	655664577	655666606	3,104	<0,001	7
English_Rand_3	652514886	652514886	652514886	2,979	0,000	10
English_Rand_4	656764071	656764071	656764071	2,699	0,000	10
English_Rand_5	654557377	654557377	654557377	2,871	0,000	10
French	13491323058	13491323058	13491501777	19,364	0,001	6
French_Rand_1	621603962	621603962	621615197	13,378	0,002	7
French_Rand_2	624613625	624613625	624613657	14,383	<0,001	9
French_Rand_3	622277343	622277343	622278607	13,796	<0,001	8
French_Rand_4	622715248	622715248	622715905	13,650	<0,001	9
French_Rand_5	621554539	621554539	621558662	13,935	<0,001	8
Italian	4025754990	4025754990	4025754990	8,746	0,000	10
Italian_Rand_1	2626963740	2626963740	2626963740	7,459	0,000	10
Italian_Rand_2	2632639438	2632639438	2632639438	8,066	0,000	10
Italian_Rand_3	2623137727	2623137727	2623137727	7,954	0,000	10
Italian_Rand_4	2629747342	2629747342	2629747342	8,252	0,000	10
Italian_Rand_5	2626060706	2626060706	2626060706	8,294	0,000	10
Spanish	1229359070	1229359070	1229359070	8,364	0,000	10
Spanish_Rand_1	823360277	823360277	823360277	5,560	0,000	10
Spanish_Rand_2	826559542	826559542	826559542	5,911	0,000	10
Spanish_Rand_3	822666809	822666809	822666809	5,984	0,000	10
Spanish_Rand_4	828720885	828720885	828720885	6,003	0,000	10
Spanish_Rand_5	824672158	824672158	824672158	5,503	0,000	10

4.5 Resultados para instâncias da língua portuguesa

A Tabela 6 reúne os resultados obtidos pelo ILS-SKQAP, com a parametrização selecionada de $I_{MS} = 15$ e $I_{ILS} = 4n$, para as instâncias da língua portuguesa desenvolvidas neste trabalho.

Pode-se perceber que o ILS-SKQAP também proporcionou bons resultados para as instâncias da língua portuguesa, alcançando a melhor solução em todas elas e obtendo *gaps* médios menores que 0,005%, com tempos computacionais de 16,78 segundos, em média, para esta classe. Nota-se que o tempo computacional médio obtido para as instâncias em português é similar ao das instâncias em francês, dado que possuem tamanhos similares (41 para French, 38 para French_Rand_1 a 5 e 40 para as 6 instâncias em português).

A Figura 35 mostra o gráfico de convergência empírica do ILS-SKQAP para a instância Portuguese, em cuja geração foi utilizado método similar ao apresentado para as instâncias existentes. Pode-se observar que, para essa instância, o algoritmo encontrou a melhor solução, com probabilidade de mais de 80%, em torno de 30 segundos.

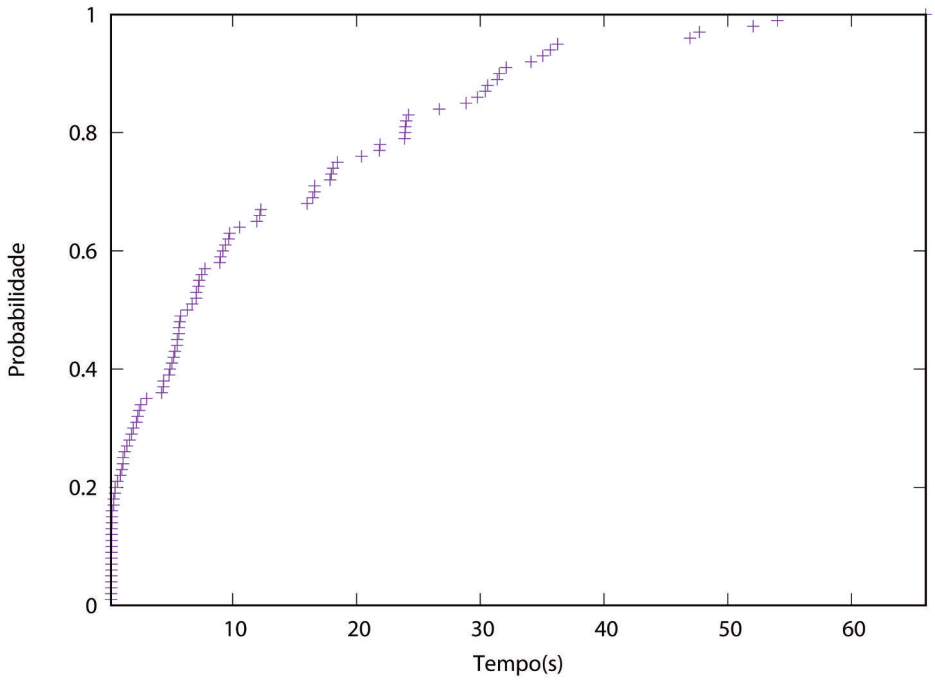


Figura 35: Gráfico de convergência para Portuguese

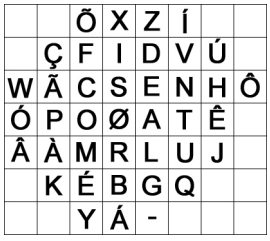
Os melhores *layouts* obtidos para as instâncias da língua portuguesa estão ilustrados na Figura 36 de (a) a (f), destacando-se o *layout* correspondente à instância Portuguese, baseada no corpus brasileiro.

Tabela 6: Resultados do ILS-SKQAP por instância da língua portuguesa

Instance	**ILS-SKQAP**	**Avg. Sol.**	**Avg. Time (s)**	**Avg. Gap (%)**	**#Best**
Portuguese	1123847642543	1123847642543	16,812	0,000	10
Portuguese_Rand_1	157338467337	157338467337	16,555	0,000	10
Portuguese_Rand_2	153688782196	153695204883	16,592	0,004	6
Portuguese_Rand_3	185760833068	185764279729	17,762	0,002	6
Portuguese_Rand_4	155889624542	155889624542	16,514	0,000	10
Portuguese_Rand_5	162264039418	162268123369	16,457	0,003	3

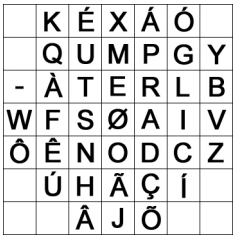
Portuguese

(a)



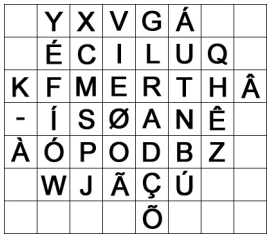
Portuguese_Rand_1

(b)



Portuguese_Rand_2

(c)



Portuguese_Rand_3

(d)

Portuguese_Rand_4

(e)

Portuguese_Rand_5

(f)

Figura 36: Melhores *layouts* para as instâncias da língua portuguesa

Capítulo 5

Considerações Finais e Trabalhos Futuros

Este trabalho aborda o *single-finger keyboard layout problem* (SK-QAP), originalmente proposto por Dell'Amico et al. (2009). Uma revisão de trabalhos relacionados foi conduzida, considerando não só o problema de desenvolvimento de *layout* de teclado para um dedo, como também foram incorporados estudos que discutem o problema para teclados destinados a n dedos. Decidiu-se expandir o escopo da literatura devido ao número restrito de trabalhos que tratam do SK-QAP utilizando uma abordagem em Pesquisa Operacional.

Estabeleceu-se o uso da meta-heurística ILS como método para resolver o SK-QAP devido à sua simplicidade de atuação. O algoritmo proposto foi denominado ILS-SKQAP e inicia-se com um procedimento construtivo constituído de componentes gulosos e aleatórios. Na fase de busca local, o ILS-SKQAP usa o procedimento RVND, composto por três estruturas de vizinhanças: *pairwise-exchange* ($N^{(1)}$), *contour filling* ($N^{(2)}$) e *two pairs swap* ($N^{(3)}$). As primeiras duas estruturas foram apresentadas no trabalho de Dell'Amico et al. (2009) e a última foi desenvolvida no presente trabalho. Para escapar de ótimos locais, duas estratégias de perturbação são incorporadas ao ILS-SKQAP: *ejection chain* ($P^{(1)}$) e *multiple pairwise-exchange* ($P^{(2)}$).

O ILS-SKQAP foi testado nas 24 instâncias propostas por Dell'Amico et al. (2009) para os idiomas inglês, francês, italiano e espanhol, mostrando-se eficiente na resolução do SK-QAP ao proporcionar resultados extremamente competitivos, registrando *gaps* médios inferiores a 0,001% por classe de instância, com tempos computacionais, em média, inferiores a 20 segundos. Na maioria das execuções do algoritmo, a melhor solução reportada das instâncias existentes foi encontrada.

Este trabalho apresenta 6 novas instâncias para o SK-QAP, adicionando o português à lista de idiomas abordados para este problema. O algoritmo ILS-SKQAP foi capaz de resolvê-las, gerando um *layout* para usuários de teclados de língua portuguesa, a partir

da instância real Portuguese.

5.1 Estratégias não satisfatórias

Duas estratégias foram tentadas, neste trabalho, para abordar o SK-QAP de maneira diferenciada, almejando facilitar a resolução de instâncias mais difíceis, como as da língua francesa, e melhorar a qualidade das soluções obtidas pelo ILS-SKQAP. A primeira delas consistiu em armazenar, a cada chamada do procedimento RVND, as linhas da solução obtida num *pool*, juntamente com seus custos individuais e suas posições relativas na solução. Um modelo matemático foi desenvolvido para encontrar um conjunto de linhas que retornasse o menor valor para a função objetivo, sendo resolvido através do CPLEX. No entanto, esse método resultou apenas no retorno da melhor solução obtida pelo algoritmo, sem gerar melhoras adicionais.

A segunda estratégia testada para melhorar as soluções obtidas pelo ILS-SKQAP foi através da movimentação horizontal de linhas do teclado, modificando a disposição das teclas e os valores da matriz D. Para isso, a cada execução do RVND, a solução obtida foi armazenada e testaram-se novas posições horizontais para as linhas do *layout* obtido. Entretanto, devido à natureza do algoritmo de alocar caracteres de maior fluxo entre eles na menor distância possível, o ganho obtido ao diminuir a distância entre duas teclas que contêm caracteres com menor fluxo associado não superou as perdas ao, consequentemente, aumentar a distância entre outras duas teclas contendo caracteres com maior fluxo associado.

5.2 Trabalhos futuros

Baseando-se nas estratégias não satisfatórias, outra maneira de abordar o SK-QAP seria através da movimentação horizontal das linhas realizada anteriormente à alocação dos caracteres, de forma a tirar proveito de menores distâncias na disposição inicial de teclas onde pares de caracteres com maior fluxo entre eles seriam alocados. Outras disposições de teclas, em formatos diferenciados, também podem ser testadas na tentativa de melhorar o custo das soluções obtidas.

Este estudo tem como foco os aspectos teóricos da utilização de um teclado para um dedo. Para trabalhos futuros, sugere-se desenvolver experimentos práticos com usuários que escrevem nos idiomas abordados pelas instâncias do SK-QAP, de maneira a avaliar suas experiências em aspectos como tempo e esforço para aprender um novo *layout*, velocidade e taxa de erros de digitação, conforto ao digitar, entre outros aspectos que quantifiquem melhor as vantagens e desvantagens associadas à adoção de um novo *layout*.

Novas instâncias para outras línguas também devem ser desenvolvidas para ampliar a variedade de *layouts* otimizados disponíveis ao público.

Referências

AIEX, R. M.; RESENDE, M. G. C.; RIBEIRO, C. C. Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, v. 1, n. 4, p. 355–366, Sep 2007. ISSN 1862-4480. Disponível em: <https://doi.org/10.1007/s11590-006-0031-4>.

ALSWAIDAN, N.; HOSNY, M. I.; NAJJAR, A. B. A genetic algorithm approach for optimizing a single-finger arabic keyboard layout. In: ARAI, K.; KAPOOR, S.; BHATIA, R. (Ed.). *Intelligent Systems in Science and Information 2014*. Cham: Springer International Publishing, 2015. p. 261–277. ISBN 978-3-319-14654-6. Disponível em: <https://doi.org/10.1109/SAI.2014.6918206>.

BEHESHTI, Z.; SHAMSUDDIN, S. M. A review of population-based meta-heuristic algorithm. v. 5, p. 1–35, mar 2013. Disponível em: <https://www.researchgate.net/publication/270750820_A_review_of_population-based_meta-heuristic_algorithm>.

BI, X.; SMITH, B. A.; ZHAI, S. Quasi-qwerty soft keyboard optimization. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2010. (CHI '10), p. 283–286. ISBN 978-1-60558-929-9. Disponível em: <http://doi.acm.org/10.1145/1753326.1753367>.

BI, X.; SMITH, B. A.; ZHAI, S. Multilingual touchscreen keyboard design and optimization. *Human-Computer Interaction*, Taylor & Francis, v. 27, n. 4, p. 352–382, 2012. Disponível em: <http://www.tandfonline.com/doi/abs/10.1080/07370024.2012.678241>.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 35, n. 3, p. 268–308, 2003. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/937503.937505>.

BOLUFÉ-RÖHLER, A.; CHEN, S. Minimum population search - lessons from building a heuristic technique with two population members. In: *2013 IEEE Congress on Evolutionary Computation*. [s.n.], 2013. p. 2061–2068. ISSN 1089-778X. Disponível em: <http://ieeexplore.ieee.org/document/6557812/>.

BURKARD, R.; RENDL, F. A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, v. 17, n. 2, p. 169 – 174, 1984. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/0377221784902315>.

BURKARD, R. E. Quadratic assignment problems. In: _____. *Handbook of Combinatorial Optimization*. New York, NY: Springer New York, 2013. p. 2741–2814. ISBN 978-1-4419-7997-1. Disponível em: <https://doi.org/10.1007/978-1-4419-7997-1_22>.

BURKARD, R. E.; KARISCH, S. E.; RENDL, F. Qaplib - a quadratic assignment problem library. *Journal of Global Optimization*, v. 10, n. 4, p. 391–403, jun 1997. ISSN 1573-2916. Disponível em: <https://doi.org/10.1023/A:1008293323270>.

CRAMA, Y.; KOLEN, A. W. J.; PESCH, E. J. Local search in combinatorial optimization. In: _____. *Artificial Neural Networks: An Introduction to ANN Theory and Practice*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995. p. 157–174. ISBN 978-3-540-49283-2. Disponível em: <https://doi.org/10.1007/BFb0027029>.

DELL'AMICO, M. et al. The single-finger keyboard layout problem. *Computers & Operations Research*, Elsevier Science Ltd., Oxford, UK, UK, v. 36, n. 11, p. 3002–3012, nov 2009. ISSN 0305-0548. Disponível em: <http://dx.doi.org/10.1016/j.cor.2009.01.018>.

DORIGO, M.; STÜTZLE, T. Ant colony optimization: Overview and recent advances. In: _____. *Handbook of Metaheuristics*. Boston, MA: Springer US, 2010. p. 227–263. ISBN 978-1-4419-1665-5. Disponível em: <https://doi.org/10.1007/978-1-4419-1665-5_8>.

DUNLOP, M.; LEVINE, J. Multidimensional pareto optimization of touchscreen keyboards for speed, familiarity and improved spell checking. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2012. (CHI '12), p. 2669–2678. ISBN 978-1-4503-1015-4. Disponível em: <http://doi.acm.org/10.1145/2207676.2208659>.

EBERHART, R.; KENNEDY, J. A new optimizer using particle swarm theory. In: *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*. [S.l.: s.n.], 1995. p. 39–43.

EGGERS, J. et al. Optimization of the keyboard arrangement problem using an ant colony algorithm. *European Journal of Operational Research*, v. 148, n. 3, p. 672 – 686, 2003. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221702004897>.

FITTS, P. M. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, v. 47, p. 381–391, jul 1954. Disponível em: <https://www.researchgate.net/publication/10366791>.

FOCACCI, F.; LABURTHE, F.; LODI, A. Local search and constraint programming. In: _____. *Constraint and Integer Programming: Toward a Unified Methodology*. Boston, MA: Springer US, 2004. p. 293–329. ISBN 978-1-4419-8917-8. Disponível em: <https://doi.org/10.1007/978-1-4419-8917-8_9>.

FRIEZE, A. et al. Algorithms for assignment problems on an array processor. *Parallel Computing*, v. 11, n. 2, p. 151 – 162, 1989. ISSN 0167-8191. Disponível em: <http://www.sciencedirect.com/science/article/pii/0167819189900252>.

GAMBARDELLA, L. M.; TAILLARD, É. D.; DORIGO, M. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, v. 50, n. 2, p. 167–176, fev 1999. ISSN 1476-9360. Disponível em: <https://doi.org/10.1057/palgrave.jors.2600676>.

GENDREAU, M.; POTVIN, J.-Y. *Handbook of Metaheuristics*. 2nd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 1441916636, 9781441916631.

GENDREAU, M.; POTVIN, J.-Y. Tabu search. In: _____. *Handbook of Metaheuristics*. Boston, MA: Springer US, 2010. p. 41–59. ISBN 978-1-4419-1665-5. Disponível em: <https://doi.org/10.1007/978-1-4419-1665-5_2>.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, Elsevier Science Ltd., Oxford, UK, UK, v. 13, n. 5, p. 533–549, 1986. ISSN 0305-0548. Disponível em: <http://dx.doi.org/10.1016/0305-0548(86)90048-1>.

GLOVER, F.; LAGUNA, M. Tabu search. In: _____. *Handbook of Combinatorial Optimization: Volume1–3*. Boston, MA: Springer US, 1999. p. 2093–2229. ISBN 978-1-4613-0303-9. Disponível em: <https://doi.org/10.1007/978-1-4613-0303-9_33>.

GUTIÉRREZ, E.; BRIZUELA, C. A. Ils-perturbation based on local optima structure for the qap problem. In: GELBUKH, A.; REYES-GARCIA, C. A. (Ed.). *MICAI 2006: Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 404–414. ISBN 978-3-540-49058-6.

HANSEN, P. et al. Variable neighborhood search. In: _____. *Handbook of Metaheuristics*. Boston, MA: Springer US, 2010. p. 61–86. ISBN 978-1-4419-1665-5. Disponível em: <https://doi.org/10.1007/978-1-4419-1665-5_3>.

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992. ISBN 0262082136.

IŞERI, A.; EKŞIOĞLU, M. Estimation of digraph costs for keyboard layout optimization. *International Journal of Industrial Ergonomics*, v. 48, p. 127 – 138, 2015. ISSN 0169-8141. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0169814115000657>.

Kantar IBOPE Media. *88% dos brasileiros que têm smartphone trocam mensagens instantâneas mensalmente*. 2016. `https://www.kantaribopemedia.com/`. Acessado em: 20-01-2018.

KARRENBAUER, A.; OULASVIRTA, A. Improvements to keyboard optimization with integer programming. In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA: ACM, 2014. (UIST '14), p. 621–626. ISBN 978-1-4503-3069-5. Disponível em: <http://doi.acm.org/10.1145/2642918.2647382>.

Kaspersky Lab. *Online Activity*. 2017. `https://index.kaspersky.com/metrics/onlineactivity`. Acessado em: 20-01-2018.

KENNEDY, J. Particle swarm optimization. In: _____. *Encyclopedia of Machine Learning*. Boston, MA: Springer US, 2010. p. 760–766. ISBN 978-0-387-30164-8. Disponível em: <https://doi.org/10.1007/978-0-387-30164-8_630>.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: *Neural Networks, 1995. Proceedings., IEEE International Conference on*. [S.l.: s.n.], 1995. v. 4, p. 1942–1948 vol.4.

KIRKPATRICK, S. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, v. 34, n. 5, p. 975–986, mar 1984. ISSN 1572-9613. Disponível em: <https://doi.org/10.1007/BF01009452>.

KOOPMANS, T. C.; BECKMANN, M. Assignment problems and the location of economic activities. *Econometrica*, Wiley, Econometric Society, v. 25, n. 1, p. 53–76, 1957. ISSN 00129682, 14680262. Disponível em: <http://www.jstor.org/stable/1907742>.

KOTSIREAS, I. S. Algorithms and metaheuristics for combinatorial matrices. In: _____. *Handbook of Combinatorial Optimization*. New York, NY: Springer New York, 2013. p. 283–309. ISBN 978-1-4419-7997-1. Disponível em: <https://doi.org/10.1007/978-1-4419-7997-1_13>.

LAWLER, E. *Combinatorial Optimization: Networks and Matroids*. Dover Publications, 1976. (Dover Books on Mathematics Series). ISBN 9780486414539. Disponível em: <https://books.google.com.br/books?id=m4MvtFenVjEC>.

LI, Y.; CHEN, L.; GOONETILLEKE, R. S. A heuristic-based approach to optimize keyboard design for single-finger keying applications. *International Journal of Industrial Ergonomics*, v. 36, n. 8, p. 695 – 704, 2006. ISSN 0169-8141. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0169814106000886>.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search: Framework and applications. In: _____. *Handbook of Metaheuristics*. Boston, MA: Springer US, 2010. p. 363–397. ISBN 978-1-4419-1665-5. Disponível em: <https://doi.org/10.1007/978-1-4419-1665-5_12>.

MACEDO, R. et al. Skewed general variable neighborhood search for the location routing scheduling problem. *Computers & Operations Research*, v. 61, p. 143 – 152, 2015. ISSN 0305-0548. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0305054815000696>.

MACKENZIE, I. S.; SELLEN, A.; BUXTON, W. A. S. A comparison of input devices in element pointing and dragging tasks. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 1991. (CHI '91), p. 161–166. ISBN 0-89791-383-3. Disponível em: <http://doi.acm.org/10.1145/108844.108868>.

MACKENZIE, I. S.; ZHANG, S. X. The design and evaluation of a high-performance soft keyboard. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 1999. (CHI '99), p. 25–31. ISBN 0-201-48559-1. Disponível em: <http://doi.acm.org/10.1145/302979.302983>.

MACKENZIE, I. S.; ZHANG, S. X.; SOUKOREFF, R. W. Text entry using soft keyboards. *Behaviour & Information Technology*, Taylor & Francis, v. 18, n. 4, p. 235–244, 1999. Disponível em: <https://doi.org/10.1080/014492999118995>.

MARTINS, S. L.; RIBEIRO, C. C. Metaheuristics and applications to optimization problems in telecommunications. In: _____. *Handbook of Optimization in Telecommunications*. Boston, MA: Springer US, 2006. p. 103–128. ISBN 978-0-387-30165-5. Disponível em: <https://doi.org/10.1007/978-0-387-30165-5_4>.

METROPOLIS, N. et al. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, v. 21, p. 1087–1092, jun. 1953. Disponível em: <http://aip.scitation.org/doi/abs/10.1063/1.1699114>.

Ministério da Cultura. *Português é hoje a quarta língua mais falada no mundo*. 2017. `http://www.cultura.gov.br/`. Acessado em: 01-03-2018.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. *Computers & Operations Research*, v. 24, n. 11, p. 1097 – 1100, 1997. ISSN 0305-0548. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0305054897000312>.

MURALI, G.; PANICKER, V. V. Optimization of a single finger keyboard layout using genetic algorithm and topsis. In: . [s.n.], 2016. v. 7, n. 2, p. 102 – 105. ISSN 2229-5518. Disponível em: <https://www.ijser.org/onlineResearchPaperViewer.aspx?Optimization-of-a-Single-Finger-Keyboard-Layout-using-Genetic-Algorithm-and-TOPSIS.pdf>.

NIKOLAEV, A. G.; JACOBSON, S. H. Simulated annealing. In: _____. *Handbook of Metaheuristics*. Boston, MA: Springer US, 2010. p. 1–39. ISBN 978-1-4419-1665-5. Disponível em: <https://doi.org/10.1007/978-1-4419-1665-5_1>.

NUGENT, C. E.; VOLLMANN, T. E.; RUML, J. An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research*, v. 16, n. 1, p. 150–173, 1968. Disponível em: <https://doi.org/10.1287/opre.16.1.150>.

OSMAN, I. H.; LAPORTE, G. Metaheuristics: A bibliography. *Annals of Operations Research*, v. 63, n. 5, p. 511–623, out 1996. ISSN 1572-9338. Disponível em: <https://doi.org/10.1007/BF02125421>.

OULASVIRTA, A. et al. Improving two-thumb text entry on touchscreen devices. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2013. (CHI '13), p. 2765–2774. ISBN 978-1-4503-1899-0. Disponível em: <http://doi.acm.org/10.1145/2470654.2481383>.

POLI, R.; KENNEDY, J.; BLACKWELL, T. Particle swarm optimization. *Swarm Intelligence*, v. 1, n. 1, p. 33–57, Jun 2007. ISSN 1935-3820. Disponível em: <https://doi.org/10.1007/s11721-007-0002-0>.

RAYNAL, M.; VIGOUROUX, N. Genetic algorithm to generate optimized soft keyboard. In: *CHI '05 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2005. (CHI EA '05), p. 1729–1732. ISBN 1-59593-002-7. Disponível em: <http://doi.acm.org/10.1145/1056808.1057008>.

REEVES, C. R. Genetic algorithms. In: _____. *Handbook of Metaheuristics*. Boston, MA: Springer US, 2010. p. 109–139. ISBN 978-1-4419-1665-5. Disponível em: <https://doi.org/10.1007/978-1-4419-1665-5_5>.

SAHNI, S.; GONZALEZ, T. P-complete approximation problems. *J. ACM*, ACM, New York, NY, USA, v. 23, n. 3, p. 555–565, jul. 1976. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/321958.321975>.

SCHEDLBAUER, M. Effects of key size and spacing on the completion time and accuracy of input tasks on soft keypads using trackball and touch input. *Proceedings of*

the Human Factors and Ergonomics Society Annual Meeting, v. 51, n. 5, p. 429–433, 2007. Disponível em: <https://doi.org/10.1177/154193120705100501>.

SCHRIJVER, A. On the history of combinatorial optimization (till 1960). In: AARDAL, K.; NEMHAUSER, G.; WEISMANTEL, R. (Ed.). *Discrete Optimization*. Elsevier, 2005, (Handbooks in Operations Research and Management Science, v. 12). p. 1 – 68. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0927050705120015>.

SOUKOREFF, R. W.; MACKENZIE, I. S. Theoretical upper and lower bounds on typing speed using a stylus and a soft keyboard. *Behaviour & Information Technology*, Taylor & Francis, v. 14, n. 6, p. 370–379, 1995. Disponível em: <https://doi.org/10.1080/01449299508914656>.

SOUZA, M. J. F. *Inteligência Computacional para Otimização*. mar 2011. Notas de aula. Disponível em: <https://www.researchgate.net/publication/267301635>.

SUBRAMANIAN, A. et al. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, Elsevier Science Ltd., Oxford, UK, UK, v. 37, n. 11, p. 1899–1911, nov. 2010. ISSN 0305-0548. Disponível em: <http://dx.doi.org/10.1016/j.cor.2009.10.011>.

TALBI, E.-G. *Metaheuristics: From Design to Implementation*. [S.l.]: Wiley Publishing, 2009. ISBN 0470278587, 9780470278581.

YAMADA, H. A historical study of typewriters and typing methods: from the position of planning japanese parallels. *Journal of Information Processing*, Information Processing Society of Japan (IPSJ), v. 2, n. 4, p. 175–202, fev 1980. ISSN 1882-6652. Disponível em: <https://ci.nii.ac.jp/naid/110002673261/en/>.

YIN, P.-Y.; SU, E.-P. Cyber swarm optimization for general keyboard arrangement problem. *International Journal of Industrial Ergonomics*, v. 41, n. 1, p. 43 – 52, 2011. ISSN 0169-8141. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0169814110001058>.

ZHAI, S.; HUNTER, M.; SMITH, B. A. Performance optimization of virtual keyboards. *Human-Computer Interaction*, Taylor & Francis, v. 17, n. 2-3, p. 229–269, 2002. Disponível em: <http://www.tandfonline.com/doi/abs/10.1080/07370024.2002.9667315>.