UNIVERSIDADE FEDERAL DO AMAZONAS

INSTITUTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

MÉTODO PARA O RECONHECIMENTO EFICAZ DE PALAVRAS EM CENÁRIOS RUÍDOSOS COMBINANDO ÍNDICES DE MOR-FRAENKEL COM HASHING PERFEITO MÍNIMO

FERNANDO ANGLADA LORES

ORIENTADOR: D. SC. MARCO ANTÔNIO PINHEIRO DE CRISTO

Manaus - AM
Abril /2018

UNIVERSIDADE FEDERAL DO AMAZONAS

INSTITUTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

MÉTODO PARA O RECONHECIMENTO EFICAZ DE PALAVRAS EM CENÁRIOS RUÍDOSOS COMBINANDO ÍNDICES DE MOR-FRAENKEL COM HASHING PERFEITO MÍNIMO

FERNANDO ANGLADA LORES

Dissertação apresentada ao Programa de Pós Graduação em Informática da Universidade Federal do Amazonas, como parte dos requisitos para a obtenção do título de Mestre em Informática, área de concentração: Sistemas digitais eye-tracking.

Orientador: D. Sc. Marco Antônio Pinheiro de Cristo.

Manaus - AM
Abril/2018

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

L869m

Lores, Fernando Anglada
Método para o reconhecimento eficaz de palavras em cenários ruídosos combinando índices de Mor-Fraenkel com Hashing Perfeito Mínimo / Fernando Anglada Lores. 2018
64 f.: il. color; 31 cm.

Orientador: Marco Antônio Pinheiro de Cristo
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. reconhecimento. 2. ruído. 3. léxico. 4. índice. I. Cristo, Marco Antônio Pinheiro de II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Método para o Reconhecimento Ágil de Palavras em Cenários Ruidosos Combinando Índices de Mor-Fraenkel com Hashing Perfeito Mínimo"

FERNANDO ANGLADA LORES

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Marco Antonio Pinheiro de Cristo - PRESIDENTE

Prof. Raimundo da Silva Barreto - MEMBRO INTERNO

Prof. David Braga Fernandes de Oliveira - MEMBRO EXTERNO

Manaus, 09 de Abril de 2018

A:

Deus, por me dar a oportunidade de viver e por estar comigo em cada passo que eu tomei, por fortalecer meu coração e iluminar minha mente e por ter posto no meu caminho as pessoas que foram meu apoio e companheiros durante todo o período de estudo.

Minha mãe Besaida Lores, por me dar vida, me amar, acreditar em mim e porque você sempre me apoiou. Muito obrigado a meu pai José Anglada pela força incondicional que me deu durante minha estadia no Brasil. Obrigado aos dois por me ajudarem neste caminho tão difícil.

Meu orientador Marco Antônio Pinheiro de Cristo. por seu grande apoio e motivação para o culminar os estudos e para a elaboração desta dissertação. Meus professores em geral da pós-graduação: David B.F. de Oliveira, Eulanda M. dos Santos, Altigran S. da Silva, Eduardo L. Feitosa e Edleno S. de Moura.

Meus amigos que nos apoiemos na nossa formação profissional e que até agora, ainda somos amigos:

AGRADECIMENTO

Em primeiro lugar, agradeço aos meus pais que fizeram todos os esforços para que eu terminasse esta etapa da minha vida, pelo apoio em todos os momentos difíceis, eles sempre estiveram perto de mim e graças a isso eu consegui conquistar mais um logro na minha vida.

Ao professor Marco Antônio Pinheiro de Cristo, pela sua orientação, total apoio, disponibilidade, pelo saber que transmitiu, pelas opiniões e críticas, total colaboração no solucionar de dúvidas, problemas que foram surgindo ao longo da realização deste trabalho e por todas as palavras de incentivo.

Agradeço a todos aqueles que sempre me apoiaram incondicionalmente, que apostaram em mim mais de que ninguém e que seguramente compartilham minha alegria: minha família e amigos.

A todos os professores do Instituto de Computação da Universidade Federal do Amazonas (UFAM-ICOMP) pelos ensinamentos e compreensão que brindaram ao longo do curso.

Epígrafe

Autor da epígrafe

RESUMO

Em sistemas de digitação com os olhos é necessário reconhecer as palavras que o usuário deseja digitar a partir dos movimentos que seus olhos fazem em um teclado virtual. Este processo pode ser visto como a transmissão de palavras através de um canal ruidoso. Assim, a tarefa de reconhecimento consiste em determinar com que palavras de um léxico se parece mais uma palavra distorcida ao passar por um canal ruidoso. Para isso são selecionadas um conjunto de palavras que possam ser transformadas na palavra de entrada mediante k operações de edição de caracteres, utilizando estimativas que envolvem o conhecimento de domínio sobre fontes de ruído e distribuições de erros para classificar os possíveis candidatos. Tais estimativas podem se tornar inviáveis dependendo do tamanho do léxico e da quantidade de ruído no cenário de interesse. Neste trabalho, atacamos este problema propondo métodos eficientes de cálculo de distância de edição usando índices de Mor-Fraenkel combinados com um hashing perfeito mínimo. Estes métodos permitem o processamento precoce da maioria dos candidatos promissores, avaliando a mínima quantidade de operações a serem feitas no processo de pesquisa, e proporcionando maior rapidez e melhor qualidade no reconhecimento de palavras.

Palavras-chave: reconhecimento, ruído, léxico.

ABSTRACT

Eye-based typing systems need to recognize the words typed by the users interpreting their eye movements on an onscreen virtual keyboard. This process can be modeled as the transmission of words through a noisy channel. Thus, recognizing a word consists on selecting from a dictionary the words which are most similar to a distorted word (the typed word) that was received using a noisy channel. To accomplish this, the system selects the set of words which can be transformed into the typed one using until k character edit operations. These operations are weighted according to the knowledge about noise sources and error distributions observed in the eye typing scenario. To get these estimates can be hardly viable for large dictionaries and very noisy scenarios. In this work, we address this problem by proposing efficient methods to estimate edit distance using Mor-Fraenkel indices combined with a minimum perfect hashing. These methods allow the early processing of promising candidates enabling faster and better word recognition.

Keywords recognition, noise, lexicon.

LISTA DE FIGURAS

Figura 1. Canal de Ruído (Shannon, 1948).....14

Figura 2.Um exemplo de fixações e sacadas. Padrão típico de movimento dos olhos durante a leitura. 22

Figura 3. A luz visível: centro da íris (vermelha), reflexo da córnea (verde), e a saída do vetor (azul). 23

Figura 4. Geração de Viziança (residuais de s e t).....26

Figura 5. Abordagem usando Trie.....28

Figura 6. Algoritmo 1(*Get-η-neigh-r*) 38

Figura 7. Algoritmo 2 (LOOKUPRESIDUALS).....39

Figura 8. Algoritmo 3 (*LHDistance*) 44

Figura 9. Algoritmo 4 (AMPH-QHD) 46

Figura 10. Abordagem usando Hashing Perfeito Mínimo.....47

Figura 11. Processamento das strings usando o dicionário 70k e τ = 64 ms.....53

Figura 12. Tamanho de índice em relação ao MRR.....54

Figura 13. Tamanho de índice em relação ao tempo.....55

LISTA DE TABELAS

Tabela 1. Alinhamento entre as strings dpuivvcjkreert e picket (caracteres nas listas de deleções exibidos em fundo cinza) 45

Tabela 2. Probabilidades de operações de edição em EYE-PAIRS.....50

LISTA DE ABREVIATURAS E SIGLAS

IPC – *Introdução à Programação de Computadores*

UFAM – *Universidade Federal do Amazonas*

IComp – *Instituto de Computação*

2DOPMM – *Modelo bidimensional oculomotor.*

AOI – *Sequência de fixações em diferentes áreas de interesse.*

Dr – *Distância entre dois conjuntos locais aleatórios.*

MF – *Modelo de Mor-Fraenfel.*

DL – *Modelo de Damerau Levenshtein.*

$\mathcal{D}$ **–** *Dicionário com strings ordenados.*

ε_{DEL} **–** *Número de exclusões no processo de edição ideal.*

ε_{INS} **–** *Número de inserções no processo de edição ideal.*

EYE-PAIRS – *Número de inserções no processo de edição ideal.*

MRR – *Média do inverso das classificações em que a palavra correta foi recuperada para um conjunto de padrões.*

τ **-** *parâmetro de tempo limite apropriado***.**

SUMÁRIO

CAPÍTULO 1 - INTRODUÇÃO.....13

1.1 Contexto.....13
1.2 Motivação e Justificativa.....16
1.3 Problema.....17
1.4 Objetivo.....18
1.5 Contribuições.....18
1.6 Organização do Trabalho.....19

CAPÍTULO 2 - REFERENCIAL TEÓRICO.....21

2.1 Eye-tracker.....21
2.2 Métricas de distância baseadas em edição.....23
2.3 Busca aproximadas.....25

CAPÍTULO 3 - TRABALHOS RELACIONADOS.....30

CAPÍTULO 4 - UM MÉTODO EFICIENTE PARA RECONHECIMENTO DE PALAVRAS.....36

4.1.Recuperação rápida usando Hashing Perfeito.....36
4.2.Heurística linear de distância: *AMPH-LHD*.....41
4.3.Heurístca Quadrática de Distância: *AMPH-QHD*.....45

CAPÍTULO 5 - EXPERIEMNTOS E RESULTADOS.....49

5.1 Trabalhos de Referência.....49
5.2 Metodologias de Avaliação.....50
5.3 Resultados.....52

CAPÍTULO 6 - CONCLUSÕES.....56

CAPÍTULO 7 - REFERÊNCIAS.....58

Capítulo 1

INTRODUÇÃO

Neste capítulo inicial será apresentado um contexto geral sobre a entrada com ruído no sistema por meio de dispositivos de comunicação eye-tracker utilizados pelos usuários; assim como a motivação, objetivos e organização do trabalho.

1.1 Contexto

A comunicação é uma das necessidades mais importantes da vida humana depois da sobrevivência física. O homem é um ser vivente e necessita da comunicação do mesmo modo que o corpo requer água e alimento para um bom funcionamento. A habilidade de comunicação interpessoal é extremamente necessária já que permite ao indivíduo o desenvolvimento das sensações de segurança, autoconfiança, firmeza, credibilidade, felicidade e enriquecimento. Por outro lado, quando essa habilidade é deficitária, ou ineficiente, o déficit pode contribuir para a deterioração da vida pessoal e comprometer também o campo profissional.

Entre outras aplicações, os dispositivos eye-trackers (Barreto, 2012) têm sido muito usados para a comunicação por meio da visão, em particular, por pessoas com graves deficiências motoras. Eles são baseados em um conjunto de tecnologias que permitem medir e registrar os movimentos oculares de um indivíduo. Por exemplo, podem ser usados para o controle da posição do ponteiro em um teclado virtual mostrado na tela do usuário, para a digitação de caracteres (Pedrosa et al.,

2015). Na prática existem diversas soluções para a digitação de caracteres por meio dos olhos usando eye-trackers. Entre elas, as baseadas em layout Qwerty são muito utilizadas, uma vez que este layout se tornou um padrão mundial e está presente em praticamente todos os computadores e dispositivos. Sua popularidade diminui o custo de aprendizado em um teclado visual.

A digitação de caracteres usando teclados virtuais controlados pelo olhar requer várias questões a serem consideradas, como o layout de teclado, tamanho da tecla, e que a ação do usuário indique a seleção de uma tecla (Morimoto et al., 2010). Quando o usuário é capaz de mover outros músculos do corpo além dos músculos do olho, a seleção pode ser realizada por meio de ações de interação alternativas, como apertar um botão físico (Pedrosa et al., 2015). Contudo, para os fins deste trabalho, consideramos a situação em que toda a interação deve se dar por meio dos olhos.

O reconhecimento de palavras é uma tarefa desafiadora em sistemas de entrada de linguagem natural caracterizados por um alto nível de ruído. Soluções efetivas podem ser benéficas para aplicações como reconhecimento de voz, digitação por pessoas com deficiências motoras (Shaun et al., 2008) e sistemas de digitação usando os olhos (Pedrosa et al., 2015). Isso pode ser tomado como um problema de envio de um sinal através de um canal de ruído.

Figura 1. Canal de Ruído (Shannon, 1948)

Na literatura, considera-se que a variável *s* representa o sinal transmitido em um meio onde existe interferência (canal de ruído) que pode danificar a informação do sinal como mostra a Figura 1. A tarefa é corrigi-la procurando uma palavra *t* que represente esse sinal de entrada. Uma estratégia usual é determinar qual palavra *t* provavelmente corresponde a uma palavra *s* em um dicionário de palavras, usando um espaço aceitável em um tempo viável. Geralmente, a probabilidade de *t* ser *s* é estimada como inversamente proporcional a quantas operações de edição (inserção, deleção, substituição, e transposição de caracteres) devem ser feitas para

transformar *s* em *t*. Devido às restrições de tempo e espaço, o espaço de pesquisa do dicionário deve ser reduzido. Como resultado, o problema torna-se selecionar um conjunto de candidatos *t*, que possam ser transformados em *s* com até *k* operações de edição, ordenando os candidatos de acordo com a estimativa de quão semelhantes eles são. A recuperação eficaz de candidatos a uma distância de até *k* operações de edição pode ser obtida pelo uso de índices, como no algoritmo Mor-Fraenkel (Mor and Fraenkel, 1982), onde são utilizados os indexes de deleção que geram *sub-string*s de vizinhanças para verificar semelhança entre *s* e *t.* Este método considera que as operações de edição só podem ser representadas apenas por exclusões. No entanto, o tamanho desses índices cresce exponencialmente com o tamanho do dicionário e *k*. Isto, na prática, limita *k* a um pequeno valor e, por conseguinte, a sua aplicação a cenários com pouco ruído, como é o caso da digitação em teclados físicos. Estimativas efetivas para determinar o quão bom é um candidato são baseadas no conhecimento de domínio sobre fontes de ruído e distribuições de erros, incluindo probabilidades de operações de edição específicas (Hanada et al., 2016). Esse conhecimento pode ser obtido ao analisar *logs* de uso ou corpora de erro.

Uma vez que esta informação está disponível, métricas de custo quadrática, como a distância mínima de Damerau-Levenshtein (Levenshtein et al, 1966), são usadas para estimar a semelhança entre as strings usando essas informações para ponderar as edições. Desta forma, é possível obter estimativas de quão provável é um conjunto de edições que transformam *t* em *s*.

O uso do algoritmo de Damerau-Levenshtein pode ser árduo em cenários muito ruidosos, onde são necessárias muitas operações de edição (*k* é alto) para transformar *s* em *t*. Em Hanada et al. (2016), os autores foram capazes de lidar com esses problemas, observando que os erros de inserção são mais prováveis em cenários ruidosos, como sistemas eye-trackers. Eles propuseram uma versão assimétrica do algoritmo Mor-Fraenkel, onde o número de deleções no padrão de busca *s* pode ser maior do que o número de exclusões na palavra do dicionário *t*. Permitir mais deleções na entrada do que no índice possibilitou lidar com mais erros sem a necessidade de aumentar o tamanho do índice. Para acelerar a geração de candidatos, eles adotaram uma estrutura de busca baseada em uma estrutura *Trie* (Fredkin,1960), que é muito eficiente para buscas de sequências de caracteres associadas com linguagem natural.

1.2 Motivação e Justificativa

Muitos usuários com desabilidades motoras necessitam de interfaces alternativas às comumente usadas. Em especial, aqueles com problemas motores severos, causados por condições como a *Esclerose Lateral Amiotrópica* ou a *Distrofia Muscular de Duchene*, só podem depender dos seus olhos para se comunicar com computadores. Uma alternativa para estes usuários são as interfaces baseadas em dispositivos de rastreamento ocular, os eye-trackers.

No processo da comunicação do usuário com os sistemas eye-trackers ocorre uma série de perdas valiosas da informação original, que podem ser irrecuperáveis e provocadas por ruídos. Os ruídos são todos os sinais indesejáveis na transmissão de uma informação, os quais dificultam a comunicação, interferem na transmissão e perturbam a recepção ou a compreensão da mesma (Damerau, 1964). Geralmente os ruídos mais comuns nesta comunicação são relacionados com:

- Processos Linguísticos: em particular, erros ortográficos.
- Layout do teclado: erros provocados pela proximidade entre teclas e outros elementos de interface.
- Sistema Visual: perdas de sinal devido a piscadas, erros sensoriais (imprecisão do sistema ocular em perceber o alvo do movimento dos olhos) e erros gerados pelo sistema moto-neural (erros no planejamento e execução do movimento ocular uma vez estabelecido o alvo do movimento).
- Eye-Tracker: perda de informação por amostragem e imprecisão do sistema.

Um sistema de reconhecimento no qual o usuário digita com os olhos deve levar em conta estas fontes de ruído. Assim, uma vez que a cadeia de caracteres de entrada *s* é gerada, ele deve ser reconhecido, independentemente dos ruídos observados, sendo traduzida para a sequência de teclas *t* que o usuário efetivamente quis digitar. Métodos tradicionalmente usados para esse problema esperam uma baixa incidência de erros em geral, com inserções e transposições de caracteres menos frequentes que deleções e substituições (Baba et al, 2012). No

caso específico de eye-trackers, esse cenário é diferente, uma vez que devido ao alto índice de ruído, espera-se que a sequência de entrada tenha muitos caracteres indesejados (inserções), que precisam ser removidos.

Esta operação de reconhecimento deve ser rápida o suficiente para não prejudicar a experiência do usuário com a interface. Isto requer métodos normalmente baseados em índices, mas que também ocupem pouco espaço, de forma que o método de entrada não seja demasiado caro para o equipamento em que ele será usado.

Finalmente, eye-trackers são hoje baratos o suficiente para serem usados de forma cotidiana. Como descrito por Barreto (2012), tais dispositivos são de grande interesse ao possibilitar a disponibilização de informação útil para a compreensão do comportamento visual. Com a diminuição do custo e o aumento da formação de profissionais nesta tecnologia, é previsível que se produza cada vez mais pesquisas. No momento, tais dispositivos ainda se encontram em estado incipiente de uso.

1.3 Problema

Dada uma string de entrada *s* e um dicionário $\mathcal{D}$, é necessário determinar a qual string *t*, do dicionário, corresponde *s*. Especificamente, é necessário: (i) escolher um subconjunto de candidatos promissores em $\mathcal{D}$ e (ii) ordená-los de acordo com o quão provável eles correspondem a *s*.

A escolha dos candidatos corresponde a determinar, de forma suficientemente eficiente em tempo e espaço, um sub-conjunto *L* de strings de $\mathcal{D}$ que provavelmente contém a palavra que o usuário quis digitar. A ordenação das strings em *L* é tal que, dadas duas strings de *L*, t_1 e t_2, se t_1 ocorre antes de t_2, então t_1 pode ser transformada em *s* por um conjunto de operações de edição (inserção, deleção e substituição de caractere além de transposição de caracteres adjacentes) mais provável que o necessário para transformar t_2 em *s*.

1.4 Objetivo

Neste trabalho, vamos propor um método eficaz para reconhecer palavras digitadas a partir de movimentos oculares, levando em consideração a informação sobre o ruído observado, em particular, no cenário de digitação com os olhos. Esse objetivo se traduz nos seguintes objetivos específicos:

1. Investigar uma estrutura de busca que seja mais ou tão rápida quanto uma Trie e que ocupe menos espaço. Problemas com o método proposto por Hanada et al. (2016) são o espaço ocupado pela *Trie* e seu tempo mínimo de busca, proporcional ao tamanho da palavra sendo procurada.
2. Propor uma técnica de estimativa em que o tempo máximo de processamento possa ser limitado a um certo valor, de forma a garantir que o tempo de espera do usuário seja pequeno o suficiente para não atrapalhar sua experiência com a interface.
3. Propor um algoritmo estimador que tire proveito das informações mantidas pela estrutura de índice usada, uma vez que esta já contem informação detalhada sobre como strings podem ser transformadas entre si por meio de operações de edição.

1.5 Contribuições

Neste trabalho, implementamos um modelo que combina índices de Mor-Fraenkel e hashing perfeito mínimo, a fim de decodificar a entrada de caracteres da melhor forma possível, tomando em consideração o tempo de execução e as operações a serem feitas para transformá-la na informação real. Podemos sumarizar nossas contribuições nos seguintes pontos:

1. Adoção de um hashing perfeito mínimo em vez de uma Trie para acelerar a estimativa das distâncias de edição. Um hashing perfeito mínimo é capaz de encontrar strings com custo *O(1)* o que permite que a probabilidade da transformação de *s* em *t* seja estimada, com ajuda de distâncias de Damerau-

Levenshtein, no menor tempo possível. Este índice proporciona ainda alto grau de compressão (Boytsov, 2011).

2. Processamento ordenado de resíduos de cadeias de padrões e cadeias de dicionário para que os pares mais promissores sejam avaliados com antecedência. Isso nos permitiu adotar estratégias de limite tempo para podar o processamento durante a recuperação de strings.

3. Proposta de uma heurística linear para combinar os modelos de Mor-Fraenkel e Levenshtein, utilizando as informações já fornecidas pelos índices complementada com simples pós-correção das operações de transposição e substituição. O índice de Mor-Fraenkel armazena informações sobre todas as formas que podem ser obtidas por deleções de caracteres de palavras de um dicionário (strings residuais), possibilitando a rápida recuperação destas strings, seus caracteres excluídos e posições de deleção. A informação no índice já indica onde casamentos de caracteres ocorrem, o que pode ser usado para auxiliar um algoritmo que estima a probabilidade de um conjunto de erros de edição.

Estes resultados foram reportados no artigo de Cristo et al. (2017), com fator de impacto *QUALIS A1*.

1.6 Organização do Trabalho

O restante deste trabalho é organizado como segue. No Capítulo 2, é apresentado o referencial teórico. No Capítulo 3, são abordados os conceitos presentes na literatura e trabalhos relacionados na área. No Capítulo 4, são apresentados os métodos propostos. No Capítulo 5, são apresentados os experimentos e resultados obtidos. Finalmente, no Capítulo 6 são apresentadas conclusões e perspectivas de trabalhos futuros.

Capítulo 2

REFERENCIAL TEÓRICO

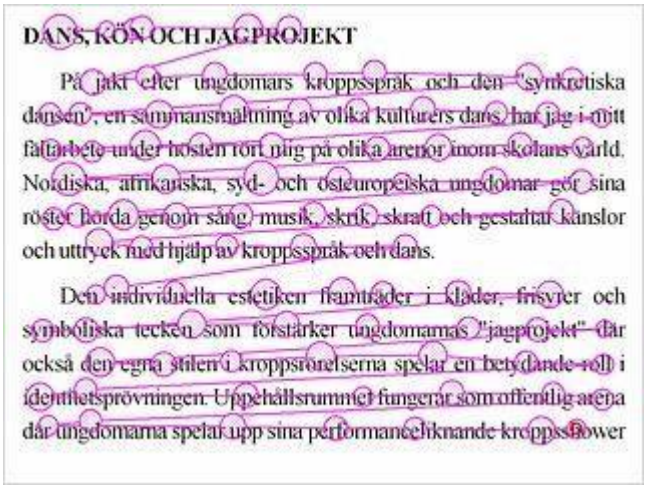
Neste capitulo é apresentado todo o referencial teórico cobrindo os dispositivos eye-traker e métodos de reconhecimento de palavras em cenários muito ruidosos.

2.1 Eye-tracker

No campo das ciências da comunicação, o eye tracking tem-se mostrado bastante versátil com aplicações em variados contextos, tais como Web, televisão, imprensa, suportes exteriores (outdoors, merchandising, decoração dos pontos de venda, etc.), videojogos, dispositivos móveis, nos eventos e sessões públicas (institucionais ou comerciais) (Liechty et al., 2003).

O conceito de eye-tracking refere-se a um conjunto de tecnologias que permite medir e registar os movimentos oculares de um indivíduo durante a amostragem de um estímulo em ambiente real ou controlado, determinando deste modo, em que área fixa está concentrada sua atenção (volume de fixações visuais gerado), por quanto tempo e que ordem segue na sua exploração visual (existência eventuais do comportamento visual) (Just et al., 1980).

Isto significa que a gravação dos movimentos oculares fornece um traçado dinâmico onde está dirigida a atenção num determinado campo visual. A medição de outros aspectos associados aos movimentos oculares, como as fixações e sacadas (cf. Capítulo 3), poderá igualmente revelar a quantidade de processamentos aplicados a objetos visualizados.


DANS, KÖN OCH JAG-PROJEKT

På jakt efter ungdomars kroppsspråk och den "synkretiska dansen", en sammansmältning av olika kulturers dans, har jag i mitt fältarbete under hösten rört mig på olika arenor inom skolans värld. Nordiska, afrikanska, syd- och östeuropeiska ungdomar gör sina röster hörda genom sång, musik, skrik, skratt och gestaltar känslor och uttryck med hjälp av kroppsspråk och dans.

Den individuella estetiken framträder i kläder, frisyrer och symboliska tecken som förstärker ungdomarnas "jag-projekt" där också den egna stilen i kroppsrörelserna spelar en betydande roll i identitetsprövningen. Uppehållsrummet fungerar som offentlig arena där ungdomarna spelar upp sina performanceliknande kroppsflower

Figura 2.Um exemplo de fixações e sacadas. Padrão típico de movimento dos olhos durante a leitura.

Os eye-trackers podem medir as rotações do olho de várias maneiras, esta medição se divide em três categorias: (i) medição do movimento de um objeto (normalmente, uma lente de contato especial) anexado aos olhos, (ii) rastreamento ótico sem o contato direto com o olho e (iii) medição de potenciais eléctricos que utilizam eléctrodos colocados em torno dos olhos.

A maioria dos rastreadores oculares modernos usa luz infravermelha para criar reflexos na córnea. O vetor entre o centro da pupila (detectado por um sistema de reconhecimento de imagens) e o ponto de reflexão na córnea pode ser utilizado para calcular a direção e amplitude do movimento ocular. Um procedimento de calibração simples do indivíduo é geralmente necessário antes de usar o rastreador ocular (Deubel et al., 1996). Se a iluminação é coaxial com o caminho óptico, então a luz reflete na retina criando um efeito semelhante ao brilho nos olhos vermelhos e se a fonte de iluminação é desviada do percurso óptico, em seguida, a pupila aparece escura porque a reflexão da luz na retina é dirigido para fora.

No entanto, este método apresenta algumas dificuldades. Ao usar luz visível para iluminar o olho, o eye-trackers pode causar algumas distrações para os usuários. Outro desafio com este método é o contraste da pupila que pode dificultar a detecção do centro da íris para o cálculo do vetor, também para a detecção dos movimentos é a obstrução causada pelas pálpebras, por exemplo, ao piscar.

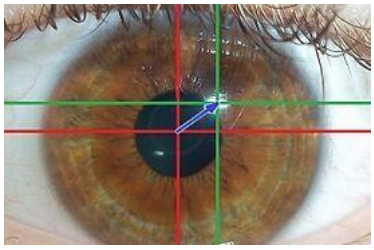


Figura 3. A luz visível: centro da íris (vermelha), reflexo da córnea (verde), e a saída do vetor (azul).

Os dispositivos para eye-tracking são muito diversos. Alguns são montados na cabeça, outros exigem que o usuário mantenha a cabeça estável (por exemplo, com um descanso no queixo), enquanto outros rastreiam o movimento do olho compensando o movimento da cabeça. A maioria usa uma taxa de amostragem de, pelo menos, 30 Hz. Embora 50/60 Hz é mais comum, hoje em dia muitos eye-tracking baseados em vídeo suportam taxas de 240, 350 ou mesmo 1000/1250 Hz, a fim de capturar movimentos de fixação ocular ou medir corretamente a dinâmica das sacadas (Rayner, 1978).

Em geral, eye-trackers especificam claramente a sua imprecisão (raio de incerteza esperada para cada ponto medido) e falta de acurácia (deslocamento esperado entre conjunto de pontos reais e medidos).

2.2 Métricas de distância baseadas em edição

Os modelos de canais ruidosos têm sido amplamente estudados, pois são utilizados em muitas aplicações para a correção ortográfica e alinhamento de sequências genéticas. Por exemplo, na correção ortográfica, a escritura errônea de uma string *w* que produz um sinal de entrada *s* pode ser vista como a transmissão de *w* através de um canal ruidoso (Kernighan et al., 1990) e (Mays et al., 1991). Para recuperar o sinal recebido *s*, o receptor tem que estimar qual é a maior probabilidade da distorção *w* entre um conjunto finito de possíveis palavras que pertencem a um dicionário ($t \in \mathcal{D}$). Isso pode ser traduzido pela equação 1 (no adiante Eq1).

$$\hat{w} = \arg\max_{t \in \mathcal{D}} P(t|s) \approx \arg\max_{t \in \mathcal{D}} P(s|t)P(t). \quad (1)$$

Na estimativa de *ŵ*, *P(t)* pode ser visto como um modelo de linguagem para *t*, e *P*(*s* | *t*) como uma métrica de distância entre *s* e *t*. As métricas mais usadas para este fim são baseadas nas operações de edição necessárias para transformar *s* em *t*. Elas são referidas como distância de Damerau-Levenshtein (Damerau, 1964,) e incluem operações de inserção, deleção, substituição e transposição. A distância básica de Damerau-Levenshtein foi ampliada várias vezes para modelar adequadamente as informações extraídas dos corpos de dados, particularmente no contexto da correção ortográfica. Este é o caso de interpretações probabilísticas da distância proposta por Church et al. (1991), Brill et al. (2000), e Norvig (2009).

Lloyd et al. (1990) propõe outra interpretação probabilística da distância, abordando o problema de encontrar o alinhamento mais provável de duas sequências genéticas. Este alinhamento consiste em fixar as posições dos caracteres comuns entre duas *strings*, como ilustrado a seguir, supondo que queremos transformar a palavra *bacon* em *aunt*:

s = b **a** co **n** λ
| |
t = λ **a** u **n** t

O alinhamento é o primeiro passo para transformar uma palavra em outra. Os caracteres comuns são alinhados como mostra o exemplo (*a*→*a* e *n*→*n*). Estes casos representam casamentos (*match*). A décima primeira letra do alfabeto grego *lambda* (*λ*) é utilizada para representar o espaço vazio, isto quer dizer que nessas posições não existe correspondência de caracteres (*b*→ *λ* e *λ* → *t*). Finalmente são feitas operações de transposição, substituição, deleção ou inserção para os outros casos (ex: *co*→ *u*). Seguindo uma interpretação da Teoria da Informação, Lloyd et al. (1990) modelaram o alinhamento de duas sequências com a hipótese de que quanto melhor é uma compressão, melhor é o alinhamento. Para calcular quão bem a cadeia é comprimida, eles derivaram as probabilidades de match, não match, inserção e exclusão de um corpus de sequências genéticas. Essa abordagem é chamada de "Comprimento Mínimo da Mensagem " (MML) (Allison et al., 1992).

Os autores em Hanada et al. (2016) utilizaram o alinhamento MML para reconhecer palavras em um sistema de entrada baseado em eye-traking. A principal mudança foi a inclusão de probabilidades de edição específicas, de forma semelhante ao que foi feito por Norvig (2009). Em particular, a distância MML foi calculada pela abordagem de programação dinâmica tradicional da seguinte forma. Dadas as strings *s* e *t*, a pontuação do script de edição ideal é obtida recursivamente a partir da pontuação de seus prefixos $s_{[1:\,i]}$ e $t_{[1:\,j]}$. A pontuação $C_{i,j}$ representa uma avaliação numérica para determinar a melhor forma de transformar duas strings, e pode ser calculada pela equação 2 (no adiante Eq2).

$$C_{0,0} = 0$$

$$C_{i,j} = min \begin{cases} C_{i-1,j} + \delta_{s_{[i]} \to \lambda} & \text{if } i > 0 \\ C_{i,j-1} + \delta_{\lambda \to t_{[j]}} & \text{if } j > 0 \\ C_{i-1,j-1} + \delta_{s_{[i]} \to s_{[i]}} & \text{if } i,j > 0, s_{[i]} = t_{[j]} \\ C_{i-1,j-1} + \delta_{s_{[i]} \to t_{[j]}} & \text{if } i,j > 0, s_{[i]} \neq t_{[j]} \\ C_{i-2,j-2} + \delta_{s_{[i-1:i]} \to s_{[i:i-1]}} & \text{if } i,j > 1, \\ & s_{[i-1:i]} = t_{[j:j-1]} \end{cases}$$

Onde o custo $\delta_{s_{[i]} \to s_{[i]}} = L_m + L_E(s_{[i]} \to s_{[i]}), \delta_{s_{[i]} \to \lambda} = L_d + L_E(t_{[j]} s_{[i]} \to t_{[j]}), \delta_{\lambda \to t_{[j]}} = L_i + L_E(t_{[j-1]} \to t_{[j-1]} t_{[j]}), \delta_{s_{[i]} \to t_{[j]}} = L_s + L_E(s_{[i]} \to t_{[j]})$, **e** $\delta_{s_{[i-1:i]} \to s_{[i:i-1]}} = L_t + L_E(s_{[i-1:i]} \to s_{[i:i-1]}). L_d, L_i, L_m,$ (2)

L_s e L_t são os valores negativos dos logaritmos das probabilidades das operações de deleção, inserção, match, substituição e transposição, e $L_E(e)$ é o valor negativo do logaritmo da probabilidade da operação específica de edição *e*.

2.3 Busca aproximadas

Muitos métodos baseados em índice foram propostos para busca aproximada. De acordo com Boytsov (2011), os mais importantes são as árvores de prefixo, os métodos de partição de padrões, os métodos de métrica espacial, os métodos de

vetores, os métodos de geração de vizinhança e as combinações desses métodos. Os métodos *Prefix Tree* (Trie) e métricas espaciais reduzem os conjuntos de palavras a serem pesquisadas, dividindo-as para criar uma hierarquia de partições de espaço que são percorridas recursivamente de acordo com uma métrica de proximidade. Os métodos de particionamento de padrões filtram strings, procurando fragmentos para poder obter um pequeno número de candidatos que são verificados, para evitar falsos positivos. Os métodos de espaço vetorial são semelhantes aos métodos de partição de padrões, exceto que eles buscam representações de vetores de cadeias em vez de fragmentos. Finalmente, os métodos de geração de vizinhança processam a cadeia de entrada aplicando todas as combinações possíveis de operações até *k* operações de edição, de modo que todas as palavras do dicionário dentro de uma distância Damerau-Levenstein de *k* sejam geradas. Para entender melhor esta abordagem propomos o seguinte exemplo:

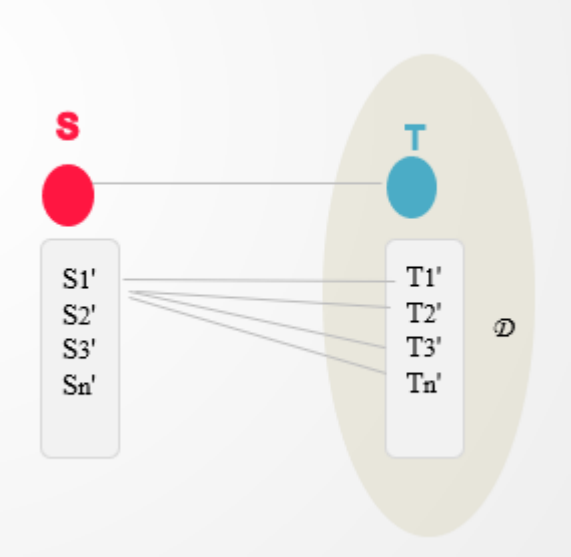


Figura 4. Geração de Viziança (residuais de s e t)

Para encontrar que palavra *t* do dicionário ($\mathcal{D}$) corresponde a uma palavra de entrada *s,* são geradas residuais de *s* e *t*, denominados vizinhanças, aplicando *k* operações de edição em *s e t* como mostra a figura 4. Se uma vizinhança de *s e* igual *a* uma vizinhança de t, possivelmente as *string* sejam iguais. Como o tamanho de *k*-vizinhança é $O(|s|^k \mid \Sigma|^k)$, principalmente devido às operações de inserção, esta abordagem é apenas prática para pequenos valores de *k*, | *s* | *e* | ∑ | (Boytsov, 2012).

Para textos em linguagem natural, de acordo com Boytsov (2012), entre todas essas famílias de métodos, o algoritmo mais eficiente em termos de tempo é o método Mor-Fraenkel (Mor et al., 1982). No entanto, é restrito a cenários com um pequeno *k* e um pequeno dicionário. O método de Mor-Fraenkel baseia-se na ideia de que todas as operações de edição em um padrão de pesquisa podem ser representadas apenas por exclusões. Isso se deve ao fato de que substituições e transposições podem ser representadas por meio de inserções e exclusões, e uma inserção no padrão de pesquisa é equivalente a uma exclusão em uma sequência do dicionário. Por exemplo, *s* = *hous* pode casar com *t* = *house* ($t \in \mathcal{D}$), inserindo um *e* no final de *s*, o que equivale a excluir um *e* do final de *t*. Assim, dada a string *t*, se soubermos todas as strings que podem ser obtidas por até *k* deleções de *t* (*a vizinhança de k deleções de t*), estas podem ser casadas exatamente com *s* para saber se *t* casa com s em até *k* deleções. Note que as vizinhanças de até *k* deleções podem ser obtidas para todas as palavras de um dicionário em tempo de indexação (off-line). Assim, são armazenadas em um índice as strings resultantes das deleções (*strings residuais* ou *resíduos*) juntamente com os seus caracteres excluídos e respectivas posições de exclusão. Este índice é chamado *índice de deleção*.

Essa ideia básica foi melhorada por muitos autores. Em particular, Bocek et al. (2007) e Garbe (2012) propuseram maneiras inteligentes de usar índices Mor-Fraenkel sem precisar armazenar as strings residuais. Boytsov (2012) propôs uma versão muito escalável usando um hashing perfeito mínimo para a pesquisa, e uma representação comprimida do índice. Chegrane et al. (2013) apresentou um algoritmo tão eficiente no tempo de pesquisa e no espaço de índices quanto o de Boytsov.

Motivados por cenários com eye-tracker, onde os erros de inserção na cadeia de padrões são mais prováveis do que outros erros, os autores em Hanada et al. (2016) propuseram uma versão assimétrica da abordagem de Mor-Fraenkel. Observaram que, embora o índice ainda seja muito caro devido à necessidade de armazenar as vizinhanças de deleção para todas as palavras do dicionário, ele pode ser viável se *k* (nas vizinhanças de deleção das palavras do dicionário) for restrito a um valor pequeno (até 3, por exemplo). Assim, eles propuseram uma abordagem assimétrica, onde o valor de *k* na vizinhança do padrão de pesquisa pode ser maior

que o das vizinhanças das palavras do dicionário. Em seus métodos, eles usam uma estrutura Trie para pesquisar os resíduos.

Dada uma correspondência entre uma sequência residual *s'*, derivada de *s*, e uma string residual *t'*, derivada de *t*, existem 4 possíveis resultados: (i) *s* foi encontrado como uma palavra de dicionário; (ii) *s* corresponde a uma palavra residual do dicionário; (iii) um resíduo de *s* corresponde a uma palavra de dicionário; e (iv) uma string residual de *s* corresponde a um resíduo de uma palavra do dicionário.

Para os casos (i), (ii) e (iii), $P(t \mid s)$ pode ser estimado linearmente. No entanto, para o caso mais comum (iv), a Eq. 2 deve ser usada. A figura 5 mostra o funcionamento dessa abordagem.

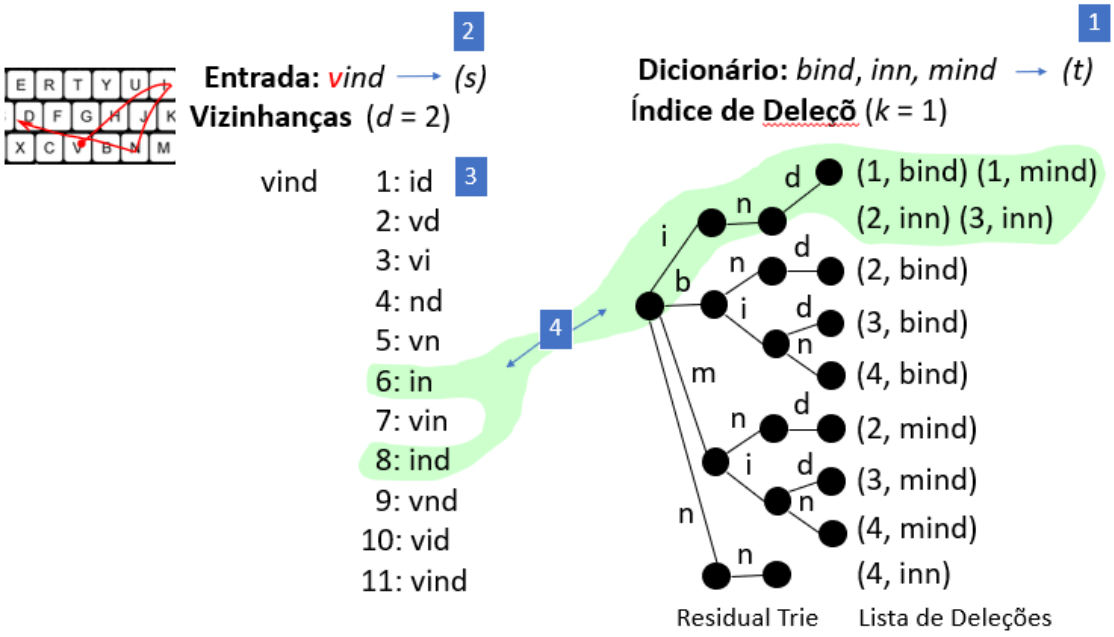


Figura 5. Abordagem usando Trie.

Como se observa na figura 5, para um dicionário de três palavras (*bind, inn, mind*) e apenas uma operação de edição de caracteres, ou seja, para *k = 1.* No passo 1, são criados os índices de deleções. Como resultado, é obtida uma lista de residuais baseada em uma estrutura Trie para cada palavra *t* do dicionário. No passos 2 e 3 o sistema recebe a palavra de entrada do usuário (ex: *vind*) e cria sua vizinhança. No exemplo, a vizinhança é criada considerando até duas deleções, ou seja, *d = 2*, o que resulta nas *strings* residuais *id, vd, vi, nd, vn, in, vin, ind, vnd, vid* e *vind*. Note que como *d* é diferente de *k*, o método é descrito como assimétrico.

Finalmente no passo 4, as strings residuais são buscadas na Trie. Para cada string encontrada, são feitas as operações de match, inserção, substituição e deleção sobre os residuais, usando a abordagem de Damerau-Levenshtein, para determinar a menor distância entre os residuais de *s* e *t.* Desta forma, é possível determinar a palavra dentro do dicionário com maior probabilidade de ser a palavra digitada pelo usuário. No exemplo em curso o sistema reconheceu que a palavra entrada pelo usuário, *vind*, corresponde às palavras do dicionário *bind*, *mind* e *inn* usando até *d* exclusões e *k* operações de edição. A estimativa de erros de edição deve ranquear palavras como *mind* e *vind* como mais prováveis que *inn*.

Ao contrário dos métodos descritos anteriormente, nossa abordagem é projetada para permitir a poda de processamento com baixo impacto na precisão e inclui uma heurística linear que suporte probabilidades de edição específicas. Além disso, nossa abordagem também difere de Hanada et al. (2016), Garbe (2012), Boytsov (2012) e Chegrane et al. (2013), porque não se restringe a uma distância clássica de Damerau-Levenshtein, senão que adapta uma abordagem de Mor-Fraenkel para senários muito ruidosos. Em comparação ao trabalho Hanada et al. (2016), além de suportar a poda e uma heurística linear, o método adota um hashing perfeito em vez de um Trie.

Capítulo 3

TRABALHOS RELACIONADOS

Neste capitulo são apresentados trabalhos relacionados referentes aos métodos de reconhecimento de palavras em cenários muito ruidosos.

Na literatura científica há diversos trabalhos relacionados ao nosso. Esses trabalhos abordam a modelagem do sistema cinético ocular humano, o aprimoramento do desempenho de dispositivos eye-trackers e a correção de erros usando modelos de ruído. Nesta seção, apresentamos alguns desses trabalhos.

O sistema cinético ocular é um dos sistemas humanos mais bem estudados na literatura científica, em função de sua importância e propriedades de fácil modelagem, como a natureza balística do movimento dos olhos em sacadas e as correlações existentes entre vários parâmetros. Enderle et al. (2000), apresentam uma visão geral de modelos de controle do movimento ocular propostos na literatura de engenharia bio-médica. O foco deste trabalho foi revisar os modelos matemáticos do sistema de movimento ocular rápido (as sacadas) e sua estratégia de controle. A literatura sobre o sistema de movimento ocular rápido é muito grande e, portanto, esta revisão não é exaustiva, mas sim uma amostra representativa do campo dos sistemas oculomotor. Esses sistemas respondem a estímulos visuais e auditivos, resultando em movimentos oculares como: movimentos rápidos, suaves, vestibulares, de convergência e optocinéticos. Cada um desses movimentos é controlado por um sistema neural diferente e todos compartilham o mesmo caminho comum. Características importantes das sacadas foram determinadas como um meio de avaliar a qualidade dos modelos sacádicos; assim como o aumento crescente da complexidade dos modelos foram apresentados com o objetivo final de construir um modelo de sacada de tipo homeomorfo da planta oculomotor. Estes

modelos permitem a análise da estática e dinâmica de propriedades dos músculos oculares. Finalmente, o monitoramento das sacadas foi considerado a partir da base de monitorar a teoria de sistemas e considerações anatômicas.

Outros trabalhos nesta linha são os de Beers (2007 e 2010) que focam, contudo, nos ruídos observados nesses modelos. O intuito desses trabalhos foi caracterizar os ruídos do sistema e determinar o princípio de controle em execução pelo sistema neuro motor. O autor apresentou evidências que sugerem que o sistema controla os movimentos com o intuito de minimizar os erros de planejamento e execução. Nenhum destes trabalhos é direcionado ao uso de eye trackers, embora os utilizem para a coleta de dados.

Komogortsev et al. (2009) abordam o problema do ponto de vista da interação do ser humano como o computador (HCI), focando seu estudo em sistemas que usam o movimento ocular como um método de entrada de dados. Diferente dos trabalhos anteriores, os modelos propostos não buscam a representação do sistema visual para a sua melhor compreensão. O autor está mais interessado em compensar deficiências da entrada com eye trackers através da previsão das trajetórias de movimento dos olhos. O trabalho apresenta um modelo matemático para eye-tracker, que usa propriedades anatômicas do sistema visual para prever trajetórias do movimento ocular em três dimensões. Ele faz várias simplificações nos modelos anteriores, evitando integrais de primeira, segunda e terceira ordens, com intuito de obter uma simulação de tempo real. O modelo é transformado em uma forma de filtro de Kalman (Komogortsev, 2007) para fornecer sinal de previsão da posição contínua dos olhos durante todos seus movimentos, fazendo uso das propriedades do controle cerebral para as transições entre movimentos rápidos (sacadas) e lentos (fixações, perseguição) dos mesmos. Comparado com um modelo de estado de Kalman, ele melhora o reconhecimento em torno de 2-3% (Komogortsev et al., 2008). Resultados obtidos com o modelo proposto demostraram uma maior acurácia na precisão do movimento dos olhos, previsão e capacidade de desempenho em tempo real.

Outra linha de trabalhos relacionados é aquela que foca no projeto e avaliação de interfaces para os olhos, em particular, teclados virtuais. Por exemplo, o trabalho de Zhai e Kristensson (2012), que investigou quão rápido pode ser uma entrada em um teclado visual QWERTY. A ideia é aproveitar a alta redundância de linguagens naturais para permitir que os usuários simplesmente olhem suas letras

desejadas sem deter-se em cada uma (modelo este conhecido como dwell-free), como em um teclado de gestos de telefones móveis. Foi criado um sistema que simula um reconhecedor perfeito para dwell-free que foi utilizado para investigar quão rápido os usuários podem escrever usando o dispositivo. Resultados demostraram que depois de 40 minutos de prática, os usuários alcançaram um ritmo médio de 46 palavras por minuto. Isso corresponde ao dobro de velocidade obtida com métodos atuais de digitação pelo movimento ocular, normalmente baseados normalmente em tempos de espera por tecla (dwell-based). Um modelo da atuação humana demonstra que sistemas de digitação com o olho tradicionais, dwell-based, são altamente improváveis e jamais atingiram uma estimativa razoável de rendimento.

Outra abordagem proposta na literatura foi apresentada por Hoppe et al. (2013). Nela, o eye-tracker usa os traços que correspondem ao scanpath (caracteres capturados pelo eye-tracker no processo de digitação) para compará-los com os traços esperados de palavras no dicionário. Para tanto, é usado o algoritmo de Needleman-Wunsch. Mesmo que a abordagem ainda tenha alguns inconvenientes, como o problema de primeira letra (ning), os autores argumentaram que ela pode ser uma valiosa contribuição para os investigadores da área, pois mostram seu uso em problemas desse tipo. Um método relacionado com este último foi proposto anteriormente por Jarodzka et al. (2010). Neste estudo, os scanpaths são vistos como sequências de vetores geométricos que corresponde a sacadas subsequentes. A representação do vetor mostra o comprimento da direção de cada sacada. A mesma é definida por uma posição de partida (fixação N) e de término posição (fixação N + 1). Nesta representação são preservadas várias propriedades, tais como a forma, o comprimento, a posição e a duração das fixações.

Pedrosa et al. (2015) propõe uma técnica de digitação dwell-free onde os caracteres digitados com os olhos são filtrados de forma a produzirem candidatos presentes em um dicionário. Os vários candidatos obtidos são classificados com base no seu tamanho e frequência de uso no idioma, para compor uma lista de sugestões aos usuários. Usando esta técnica, os pesquisadores mostraram que apesar da necessidade de selecionar cada palavra que digitam, os usuários atingem velocidades altas, da ordem de 15.95 palavras por minuto, após 100 min de digitação.

Outra linha de trabalhos relacionados com o nosso são os voltados para a correção ortográfica. Esta é uma área extensa com um grão número de contribuições ao longo do tempo. Para uma revisão recente da literatura, recomendamos o trabalho de Boytsov (2011). Neste trabalho, são apresentados os conceitos básicos relacionados às técnicas usadas em correção ortográfica, o que inclui (i) métricas para o cálculo da distância de edição entre strings, como Damerau-Levenshtein, e algoritmos eficientes para obtê-las; (ii) heurísticas para seleção e redução de candidatos, como compressão de alfabetos e usos de limites matemáticos conhecidos para o problema; e (iii) estruturas e algoritmos eficientes para busca em dicionário como tries, índices de sufixos, hashing perfeito e índices de deleção.

Os métodos apresentados são caracterizados pela identificação de até *k* operações de edição necessárias para transformar uma palavra entrada em uma de um dicionário. Em geral, as operações de edição consideradas são a inserção, a deleção, a substituição e a transposição de caracteres. A maioria dos métodos trata todas essas operações como de mesma importância. Em termos práticos, esses métodos podem ser usados para valores de *k* até 4, o que é razoável, já que erros de edição são raros na maioria dos cenários em que esses métodos são usados. Entre os métodos apresentados, o mais eficiente foi o Mor-Fraenkel. Este método é baseado no uso de um índice de deleções ou de resíduos, ou seja, palavras resultantes de remoções de até *k* caracteres das palavras do dicionário. Dada a palavra de entrada, seus resíduos são procurados no índice de resíduos (previamente construído) e, caso encontrados, as posições das remoções dos resíduos são analisadas para determinar a distância de edição exata.

Outra revisão da área é apresentada por Jurafsky (2000), mas com um foco em métodos baseados em canais de ruído. Nestes métodos, a intuição é que o sinal original pode ser recuperado do sinal ruidoso se forem usadas estimativas razoáveis sobre as transformações que o sinal pode ter sofrido dados os ruídos característicos do canal. Norvig (2009) propõe um método baseado em canal de ruído em que a probabilidade de cada edição individual é estimada a partir de um corpus de erros comuns no idioma. Desta forma, são naturalmente capturadas diferentes fontes de ruído derivadas do uso do idioma, layouts de teclado, similaridades fonéticas, a evolução natural do idioma, contextos, etc. O método, contudo, não foi proposto para cenários com $k > 3$ (comparações de cadeias de genes, por exemplo) ou dicionários

de milhões de palavras (característicos de sites de busca). Estas limitações levaram Garbe (2012) a estender as ideias de Norvig (2009), propondo o método chamado SymSpell, uma variante do método de Mor-Fraenkel.

Ao contrário das ideias de Norvig (2009), o método SymSpell utiliza a distância de Damerau-Levenshtein, ou seja, trata cada edição possível como igualmente provável. O método consiste em otimizar a busca por evitar a aplicação de operações de edição mais caras como a inserção e substituição e reduzir o número de candidatos a serem comparados. Para evitar inserções, substituições e transposições, são obtidas todas as possíveis palavras derivadas da string de entrada e de todas as strings do dicionário por meio de até *k* deleções. Para tanto, um índice de *k*-deleções é usado. Note que deleções nas palavras do dicionário correspondem a inserções na string de entrada. Além disso, substituições e transposições podem ser vistas como pares de deleções e inserções na entrada. O erro é calculado indiretamente. O casamento entre uma palavra derivada da string original, com $D < k$ deleções, e uma palavra do dicionário tem distância D. O casamento entre a string original e uma palavra derivada de uma entrada do dicionário, com $I < k$ deleções, tem distância I. Para os demais casos onde o casamento entre uma palavra derivada da string original e uma palavra derivada de uma entrada do dicionário a distância de Damerau-Levenshtein é calculada para garantir que o limite *k* seja respeitado.

Embora SymSpell seja eficiente ainda é limitado no valor de *k*, uma vez que um *k* alto implica em um índice de deleções muito grande. Além disso, ele não usa probabilidades para cada edição específica. Um cenário em que estas limitações são importantes é o alinhamento de strings de cadeias genéticas. Neste caso, valores de $k > 4$ podem ser comuns, as strings são longas e o custo das operações variável. Por exemplo, a inserção de caracteres é normalmente mais comum que a deleção e a substituição. Como resultado, métodos têm sido propostos para lidar com probabilidades específicas por operação, como o *Minimum Message Length Optimum Alignment*, descrito em Alison et al. (1990). Nesse trabalho, os autores modelam o alinhamento mais provável entre duas cadeias genéticas, baseado nas probabilidades de operações de deleção, inserção, substituição e casamento. O modelo usa Teoria da Informação, de forma que a sequência de edições mais provável é aquela que contribui para a maior compressão do par de strings comparadas. Todos os trabalhos descritos até aqui garantem que as soluções

ótimas serão encontradas. Contudo, para longas cadeias e um grande número de candidatos, o uso de tais estratégias é inviável, o que motiva a aplicação de métodos heurísticos. Para uma revisão da literatura de sequenciamento de cadeias, recomendamos os trabalhos de Haque (2009) e Li (2010).

Finalmente, os autores em Hanada et al. (2016), propuseram uma versão do método de Mor-Fraenkel em que o número de deleções permitido na string de entrada pode ser bem maior que o número de deleções na string do dicionário. Por exemplo, 10 deleções na entrada contra 2 no dicionário. Por isso, este método é dito assimétrico. A busca nesta estrutura é feita com o auxílio de uma Trie que armazena informações para calcular rapidamente o número de operações necessárias para estimar as strings mais similares dentro de uma certa distância de edição.

Como no trabalho de Hanada et al. (2016), nosso cenário tem características similares ao do alinhamento de sequencias genéticas, uma vez que há alta incidência de erros e alguns erros são mais comuns que outros. Contudo, ele pode ser descrito como um processo tradicional de correção ortográfica, uma vez que o usuário está usando o tracker como um teclado para digitação de textos em linguagem natural. Diferente de Hanada et al. (2016), nós não usamos uma Trie como estrutura de busca e, sim, um hashing perfeito mínimo. Também modificamos nosso método para suportar corte de processamento e, assim, retornar um resultado, mesmo que parcial, dentro de um certo intervalo máximo de tempo.

Capitulo 4

UM MÉTODO EFICIENTE PARA RECONHECIMENTO DE PALAVRAS

Este Capítulo apresenta a estrutura, funcionamento e as funções implementadas dos métodos propostos.

4.1. Recuperação rápida usando Hashing Perfeito

A principal vantagem de usar um Trie para armazenar string residuais, conforme feito em Hanada et al. (2016) é não desperdiçar o tempo de processamento buscando resíduos padrões que não tenham sido analisados durante a indexação. Em um cenário de entrada de linguagem natural com muito ruído, esse recurso é importante, uma vez que muitos candidatos podem ser filtrados pela Trie, melhorando assim o custo quadrático do *MML*. No entanto, dado um padrão *s'*, o custo de pesquisa em uma Trie é proporcional ao seu tamanho, ou seja, *O*(| *s'* |). Além disso, a indexação de resíduos e as listas de deleção de Mor-Fraenkel usando uma Trie requerem muita memória. Uma alternativa para lidar com essas questões é adotar um hashing perfeito (Botelho et al., 2007) como uma estrutura de pesquisa mais eficiente, como é sugerido por Boytsov (2012). A idéia é enumerar todas as cadeias residuais *t'* do dicionário usando uma função de hash perfeita mínima *f(.)*. Como *f(.)* mapeia *m* strings para números inteiros que variam de 1 até *m*, sem colisões, o custo de pesquisa é *O(1)*. Mais especificamente, o indexador itera sobre todas as cadeias do dicionário para gerar suas vizinhanças de *k*-deleções. Como resultado, cada residual de *t* obtido aplicando até *k* deleções, está

associado a uma lista de deleção composta por triplos (t', C_t, P_t), onde $C_t = (t_{[p1]}, t_{[p2]}, ..., t_{[p\ell]})$ são os caracteres excluídos de t nas posições $P_t = (p_1, p_2, ..., p_\ell)$, com $\ell \leq k$. Cada triplo é convertido em (f (t'), C_t, P_t), de modo que eles são armazenados usando os valores da função hash perfeita como chaves. Referimo-nos ao conjunto resultante de todas as listas geradas como o índice de deleção.

Nós geramos recursivamente todos os triplos possíveis (s', C_s, P_s) em tempo de busca, onde s' é um d-residual de s, isto é, um residual obtido por exclusão de d caracteres. Um problema importante dessa estratégia, quando comparada com a geração das tripla com base em uma Trie, é a necessidade de enumerar todos os resíduos possíveis antes de avaliá-los, o que pode ser proibitivo para padrões de pesquisa muito grandes e um número elevado de deleções, d. Em uma Trie, somente os resíduos que foram analisados durante a indexação podem ser percorridos. Assim, se uma sequência de caracteres s' não foi analisada no tempo de indexação, ela não será pesquisada em nenhuma outra seqüência para a qual s' é um prefixo, o que reduz drasticamente o espaço de busca.

Uma maneira simples de lidar com essa questão é restringir a geração da vizinhança a um determinado limite de tempo. No entanto, para aplicar limites de tempo seguros, é desejável primeiro gerar os resíduos mais promissores. Em outras palavras, dado o padrão de busca s = *minds* e vizinhança de 3 deleções, preferimos primeiro combinar a residual s' = *minds*, obtida sem deleções, do que os residuais *mind* e *in* obtidos com um e três deleções, respectivamente. Assim, decidimos gerar as η-vizinhança usando uma busca em profundidade, de acordo com o algoritmo 1, onde são geradas as vizinhanças por meio da função recursiva *Get-η-neigh-r*. Dado uma string w, todos os seus resíduos w' são gerados uma única vez (linhas 21-23), juntamente com seus conjuntos correspondentes de caracteres e posições excluídos, $C^{w'}$ e $P^{w'}$ (linha 24).

```
1:  function GET-η-NEIGHBORHOOD(w, η)
2:      w: target word for η-neighborhood generation
3:      η: the max number of deletion to perform
4:      p: starting position on w
5:      for d from 0 to η do
6:          Create empty deletion index I_d
7:      add w to I_0                ▷ w: single w residual with 0 dels
8:      I_0[w] ← []                 ▷ so, its deletion list is empty
9:      C^w ← []                    ▷ deleted chars in current del list
10:     P^w ← []                    ▷ ... and its positions
11:     O^w ← (1, ..., |w|)         ▷ original positions of w chars
12:     d ← 0                       ▷ #deletions already performed
13:     if η > 0 then
14:         return GET-η-NEIGH-R(w, O, I, C^w, P^w, d, η, 0)
15:     else
16:         return I
17:
18: function GET-η-NEIGH-R(w, O, I, C^w, P^w, d, η,p)
19:     d ← d + 1
20:     if |w| > 1 then
21:         for i from p to |w| do
22:             w' ← w_[0:i] + w_[i+1:]
23:             add (C^w + w_[i], P^w + O_i) to I_d[w']
24:             if d < η then
25:                 k ← 1
26:                 for j from 1 to |w| do
27:                     if i ≠ j then
28:                         newO_k ← O_j
29:                         k ++
30:                 GET-η-NEIGH-R(w', newO, I, C^w + w_[i], P^w +
                    P[i], d, η, i)
31:     return I
```

Figura 6. Algoritmo 1(*Get-η-neigh-r*)

O algoritmo é então chamado recursivamente para gerar os resíduos de *w* (linhas 25-31). Observa-se, na linha 24, que o índice de exclusão $\boldsymbol{I}$ é criado de tal forma que $\boldsymbol{I}_d$ se refere a todos os resíduos gerados pela aplicação de *d* exclusões. Assim, para definir o conjunto de palavras candidatas dentro de um limite de tempo τ, iteramos através de resíduos *s'* obtido do padrão de entrada, ordenados de acordo com o conjunto de deleções que foram usadas para gerar os mesmos, como é mostrado no algoritmo 2.

```
1:  function LookupResiduals(s, I, τ)
2:      s: search pattern
3:      I: deletion index corresponding to all the dictionary word
        residuals t'
4:      τ: lookup timeout
5:      S ← []                          ▷ list of suugestions
6:      N ← Get-η-neighborhood(s, d, τ)
7:      for dels from 0 to d + 1 do
8:          for each s' in N_dels do
9:              L_s' ← N_dels[s']       ▷ del lists for residuals of s
10:             L_t' ← I[f(s')]         ▷ del lists for t', f(s') = f(t')
11:             for each (f(s'), C^s', P^s') in L_s' do
12:                 for each (f(t'), C^t', P^t') in L_t' do
13:                     if current running time > τ then
14:                         return S
15:                     t ← reconstruct(s', C^t', P^t')
16:                     if first time in L_t' and t ∉ D then
17:                         break
18:                     if t ∈ S then
19:                         break
20:                     S ← Evaluate(s, t, S)
21:     return S
22:
23: function Evaluate(s, t, S)
24:     score ← MML(s, t)
25:     add (t, score) to S
26:     return S
```

Figura 7. Algoritmo 2 (LOOKUPRESIDUALS)

Para conseguir isso, o algoritmo 2 implementa uma abordagem assimétrica combinada com um hash perfeito mínimo. Assim, dado um padrão de busca *s*, ele é capaz de classificar um conjunto de candidatos $t_i \in \mathcal{D}$ (dicionário ordenado), de modo que o número de deleções entre *s* e *ti* que definimos como $\boldsymbol{\varepsilon}_{DEL}(s,t_i)$ seja menor que o número máximo de deleções ($\boldsymbol{\varepsilon}_{DEL}(s,t_i) \leq d$), e o número de inserções entre eles que definimos como $\boldsymbol{\varepsilon}_{IST}(s,t_i)$ seja menor que o máximo número de operações ($\boldsymbol{\varepsilon}_{IST}(s,t_i) \leq k$), com *d* provavelmente maior que k. Primeiro são geradas as vizinhanças de *d*-deleções de *s* (linha 6). *N* fornece o índice de exclusão de todas as strings *d-residuais* de *s'* ordenado pelo número de exclusões aplicadas. A iteração é feita em (linhas 8-20), recuperando do dicionário todos os triplos $(f(t'), C^t, P^t)$, de modo que $f(s') = f(t')$ (linha 10). Se esse processo demorar mais de τ milissegundos, apenas as sugestões calculadas até o momento são retornadas (linhas 13-14).

Observe-se que $f(s')=f(t')$ implica que $s'=t'$ somente se s' for parte do conjunto de string do dicionário de residuais calculadas no tempo de indexação, de acordo com a definição de uma função hash perfeita. Se este não for o caso, $s' \neq t'$ e o par (s', t') é um falso positivo. Para detectar isso, é necessário reconstruir a string t, associada a $f(s')$, inserindo em s', nas posições $P^{t'}$ os caracteres em $C^{t'}$. Se t não for encontrado em $\mathcal{D}$, ele é reconhecido como um falso positivo e nenhum dos triplos $(f(s'), C^{t'}, P^{t'})$, $f(s') = f(t')$, pertence ao dicionário (linhas 15-17). Embora seja custosa, é preciso realizar essa reconstrução apenas uma vez por cada string de dicionário residual t'. Se $f(s') = f(t')$ e este não é um caso falso positivo, t pode ser reconstruído como foi descrito anteriormente. Dados s e t, se t ainda não foi incluído na lista de sugestões (linhas 18-19) é possível estimar a distância entre eles usando MML (linha 20).

Em nossa implementação, as funções de hash perfeitas são computadas usando minperf[3], uma biblioteca de Java capaz de gerar simultaneamente funções de hash perfeitas mínimas que precisam de menos de 1,58 bits por chave. A partir de agora, nos referimos a nosso método assimétrico como Mor-Fraenkel com hash perfeito mínimo e métrica MML (*AMPH-MML*).

O algoritmo de geração de vizinhança (algoritmo 1), gera todas as combinações possíveis para d exclusões em uma string s recursivamente, e assim gera $\binom{|s|}{d}$ string residuais em η-vizinhança. No pior caso, onde todos esses resíduos correspondem às entradas no índice com o maior número de tuplas residuais $\mathcal{L}_{max}$, o limite superior de nossa abordagem é $O(\binom{|s|}{d}|\mathcal{L}_{max}$ $||s|^2)$, onde $|s|^2$ é o custo da pontuação da avaliação de MML de um par candidato (s', t').

A sobrecarga do índice, gerada principalmente pelo armazenamento da tripla (t', Ct , Pt), tem o limite superior no pior dos casos em $O(\binom{\mu}{k} * N * \mu)$, onde N é o número de string do dicionário, μ o tamanho da maior string no vocabulário e $\binom{\mu}{k}$ o número de elementos de k-vizinhança das string com tamanho μ, sendo $N * \mu$ proporcional ao custo de armazenar triplos de reconstrução.

4.2. Heurística linear de distância: *AMPH-LHD*

Conforme observado no algoritmo 2, a estimativa de distância entre *s* e *t* é realizada por um algoritmo quadrático que não aproveita a informação já conhecida no índice de exclusão. Agora propomos uma heurística linear que explora essa informação.

Se as sequências de caracteres *s* podem ser transformadas em *t,* após *d* deleções em *s* e *k* deleções em *t* (d-residual *s'* é igual a k-residual *t'*), *s* e *t* podem ser reescritas com $s = s_0 s'_{[0]} \ldots s_{k-1} s'_{[k-1]} s_k$ e $t = t_0 t'_{[0]} \ldots t_{k-1} t'_{[k-1]} t_k$, respetivamente. Como $s' = t'$ e $t = t_0 s'_{[0]} \ldots t_{k-1} s'_{[k-1]} t_k$, transformar *s* em *t* corresponde a realizar transformações $s'_{[0]} \rightarrow s'_{[0],\ldots,}$ $s'_{[k-1]} \rightarrow s'_{[k-1]}$, $s_0 \rightarrow t_{0,\ldots,}$ $s_{k-1} \rightarrow t_{k-1,}$ e $s_k \rightarrow t_k$. Como um exemplo de funcionamento, suponha que *s* = *bacon* e *t* = aunt, com *s '= t' = an*. Transformar *s* em *t* corresponde a fazer as operações *b* → *λ*, *a* → *a*, *co* → *u, n* → *a,* e *λ* → *t.* Uma transformação $s_i \rightarrow t_i$ pode ser reescrita como *w* → *λ (eg, b* → *λ), λ* → *w (eg, λ* → *t),* ou v → w (eg, co → u), onde *v* e *w* são strings de modo que *v* ≠ *λ e w* ≠ *λ. A* transformação *w* → *λ* corresponde à deleção de caracteres $w_{[0]}, \ldots, w_{[|w|-1]}$ em *s*, enquanto *λ* → *w* corresponde à inserção de caracteres $w_{[0]}, \ldots, w_{[|w|-1]}$ em *s*. Ao contrário desses casos anteriores, *v* → *w* pode ter mais de um único script de edição (sequência de operações de edição), porque diferentes combinações de operações podem produzir o mesmo resultado. Por exemplo, *co* → *u* é obtido por *c* → *u, o* → *λ* ou *c* → *λ, o* → *u*. Além disso, substituições e transposições podem ser reescritas como combinações das outras operações (inserções, exclusões e coincidências). Como consequência, para encontrar o script de edição mais provável, todas as alternativas devem ser inspecionadas.

No entanto, suponha que as transposições não são permitidas e as substituições são mais prováveis do que os correspondentes pares de exclusões e inserções. Se | *w* | = | *v* |, a transformação $v \rightarrow w = v_{[0]}, \ldots, v_{[|v|-1]} \rightarrow w_{[0]}, \ldots, w_{[|w|-1]}$ corresponde a uma sucessão de correspondências ou substituições, dependendo se $v_{[i]} = w_{[i]}$ (coincidência) ou $v_{[i]} \neq w_{[i]}$ (substituição). Observe que o caso em que |*w*| ≠ |*v*| (por exemplo*, co* → *u*) ainda é ambíguo. Uma heurística simples para lidar com este caso é alinhar caracteres em excesso com *λ*. Assim, para | *v* | > |*w* |, $v_{[i]} \rightarrow w_{[i]}$ corresponde a uma combinação ou substituição se *i* <| *w* |, ou uma exclusão se i> = |

w |. Da mesma forma, para | v | < | w |, $v_{[i]} \rightarrow w_{[i]}$ corresponde a uma inserção se i> = | v |. Como consequência, $co \rightarrow u$ será interpretado como $c \rightarrow u$, $o \rightarrow \lambda$. Embora essa heurística simples possa lidar com soluções não ótimas, a perda de precisão é aceitável, como observaremos em nossa avaliação empírica.

Dado as observações anteriores e assumindo que ($s_{[i]}$, $t_{[j]}$) é o par atual de caracteres a serem alinhados, é possível uma das seguintes situações:

- $i \notin P^s \wedge j \notin P^t$: a operação de edição é casamento (match) de $s_{[i]}$ e o próximo par para inspecionar é ($s_{[i+1]}$, $t_{[j+1]}$).
- $i \notin P^s \wedge j \in P^t$: a operação de edição é uma substituição de $s_{[i]}$ por $t_{[j]}$ se $s_{[i]} \neq t_{[j]}$, caso contrário, é casamento de $s_{[i]}$. O próximo par a inspecionar é ($s_{[i+1]}$, $t_{[j+1]}$).
- $i \notin P^s \wedge j \in P^t$: a operação de edição é inserção de $t_{[j]}$ e o próximo par para inspecionar é ($s_{[i]}$, $t_{[j+1]}$).
- $i \notin P^s \wedge j \in P^t$: a operação de edição é eliminação de $s_{[i]}$ e o próximo par para inspecionar é ($s_{[i+1]}$, $t_{[j]}$).

Uma maneira simples de permitir transposições, é verificando se a operação atual não é parte de uma sequência de operações equivalente a uma transposição. Se for esse o caso, a sequência de operações é substituída pela transposição correspondente. Mais especificamente, as transposições são equivalentes a sequências compostas por: (i) duas substituições ($ab \rightarrow ba \approx a \rightarrow b, b \rightarrow a$)[4], (ii) uma exclusão, seguido de um match e uma inserção ($cba \rightarrow cba \approx ca \rightarrow c, b \rightarrow b, b \rightarrow ba$) e (iii) uma inserção, seguido de um match e uma eliminação ($cab \rightarrow cba \approx c \rightarrow cb, a \rightarrow a, ab \rightarrow a$). Operações de transposição com o contexto anterior são mostradas no algoritmo 3. A partir de agora, nos referimos a ele como *LHDistance*.

Como é visto no algoritmo 3, os parâmetros de *LHDistance* são: o padrão de entrada s com sua lista de exclusão P^s, a lista de exclusão P^t e a lista de caracteres correspondente C^t. Os índices i e j indicam o par atual de caracteres a serem alinhados. As variáveis n e m são usadas para acessar as listas de deleções. As operações de edição são armazenadas em ε (estrutura lista).

Para inferir corretamente as operações de edição, *LHDistance* controla o contexto usando a variável ρ. Em particular, ρ salva o caractere anterior, considerado correto. A variável ρ é necessária porque fornece o contexto para uma possível operação subsequente de exclusão ou inserção. O contexto inicial é

sempre o início da palavra (<). Para ilustrar o parâmetro ρ, suponha que um usuário digite a palavra *vip* com um caractere de erro *o*, produzindo *s* = *viop*. Para corrigir o erro, o caractere *o* deve ser excluído. Após essa exclusão, e seguindo Norvig (2009), o contexto para a inserção de *p* é ρ = *i*, resultando em $i \rightarrow ip$. A probabilidade resultante seria $P(i \rightarrow ip)$, ou seja, a probabilidade de inserir *p* após *i*. O parâmetro ρ depende da operação de edição inferida. Assim, são definidas nas funções *AddMatch*, *AddSubsOrTrans*, *AddDelOrTrans* e *AddInsOrTrans*.

Todos os pares ($s_{[i]}$, $t_{[j]}$) estão alinhados desde a linha 9 até 26 do algoritmo 3. Substituições, deleções e inserções são adicionadas somente a ε após terem sido verificados, para que não formem parte das operações de transposição (*AddSubsOrTrans*, *AddDelOrTrans* e *AddInsOrTrans*). Depois de determinar todas as operações de edição, a probabilidade de edição é calculada de acordo com Eq2 (linha 27). Finalmente, a função *Evaluate*, originalmente definida no algoritmo 2, é modificada para considerar a informação completa do índice Mor-Fraenkel.

A adoção de *LHDistance* em vez do *MML* original em esta abordagem tem o impacto de reduzir o custo da função de avaliação de $|s|^2$ a 2|s|. Assim, o limite superior da complexidade neste caso é $O(\binom{|s|}{d}|\mathcal{L}_{max}||s|)$. Neste texto, nos referimos ao nosso método assimétrico de Mor-Fraenkel com uma métrica de hash mínimo perfeito baseada em heurística linear como *AMPH-LHD*.

```
1: function LHDistance(s, C^t, P^s, P^t)
2:     E ← []                          ▷ List of edit operations
3:     i ← j ← 0                       ▷ Current char pos at s and t
4:     m ← n ← 0                       ▷ Current pos at P^s and C^t/P^t
5:     ρ ← <                           ▷ Previous key assumed correct
6:     β_xy ← return < if x = < else y
7:     K(e) ← return kind of edit oper. e (i, d, s, t, or m)
8:     while i < |s| do
9:         if i = P^s_m ∧ j = P^t_n then
10:            if s_[i] = C^t_n then                    ▷ Match
11:                ρ ← AddMatch(E, s_[i])
12:            else                                     ▷ Substitution
13:                ρ ← AddSubsOrTrans(E, s_[i], C^t_n)
14:            i ++, j ++, m ++, n ++
15:        else if i = P^s_m then                       ▷ Delete s_[i]
16:            ρ ← AddDelOrTrans(E, ρ, s_[i], β_ρ)
17:            i ++, m ++
18:        else if j = P^t_n then                       ▷ Insert C^t_n(= t_[j])
19:            ρ ← AddInsOrTrans(E, ρ, C^t_n, β_ρ)
20:            j ++, n ++
21:        else
22:            ρ ← AddMatch(E, s_[i])
23:            i ++, j ++
24:    while n < |C^t_n| do                 ▷ Insert remaining t chars
25:        ρ ← AddInsOrTrans(E, ρ, C^t_n, β_ρ)
26:        n ++
27:    return Σ_i L_K(E_i) + L_E(E_i)
28: function AddMatch(E, c_1)
29:    add (c_1 → c_1) to E
30:    return c_1
31: function AddSubsOrTrans(E, c_1, c_2)
32:    if E_last = c_2 → c_1 then
33:        E_last ← c_2c_1 → c_1c_2
34:    else
35:        add (c_1 → c_2) to E
36:    return c_2
37: function AddDelOrTrans(E, c_1, c_2)
38:    if E_last−1 = k → kc_2 ∧ k ≠ < ∧ c_1 ≠ c_2 then
39:        E_last−1 ← c_1c_2 → c_2c_1
40:        delete E_last
41:    else
42:        add (c_1c_2 → c_1) to E
43:    return c_1
44: function AddInsOrTrans(E, c_1, c_2)
45:    if E_last−1 = kc_2 → k ∧ k ≠ < ∧ c_1 ≠ c_2 then
46:        E_last−1 ← c_2c_1 → c_1c_2
47:        delete E_last
48:    else
49:        add (c_1 → c_1c_2) to E
50:    return c_2
51:
52: function Evaluate(s, t, C^t, P^s, P^t, S)
53:    score ← LHDistance(s, C^t, P^s, P^t)
54:    add (t, score) to S
55:    return S
```

Figura 8. Algoritmo 3 (*LHDistance*)

4.3. Heurístca Quadrática de Distância: *AMPH-QHD*

Nossa última heurística, *AMPH-QHD*, recebe como entrada as listas de deleções correspondentes às strings *s* e *t* provenientes do algoritmo 2, por meio da função *Evaluate*. Esta heurística quadrática considera que como *s* e *t* casaram, elas podem ser re-escritas e alinhadas nos caracteres de suas listas de deleções, ou seja, $s = s_0\ s'_{[0]}\ ...\ s_{k-1}\ s'_{[k-1]}\ s_k$ *e* $t = t_0\ s'_{[0]}...t_{k-1}\ s'_{[k-1]}\ t_k$, onde *s'* corresponde à lista de deleção de tamanho *k* comum às strings *s* e *t*. Por exemplo, a tabela 1 ilustra o alinhamento de *s = dpuivvcjkreert (resíduo duvvjreer) e t = plickets* (resíduo *les*), para lista de deleção *s' = pickt*, de tamanho *k* = 5.

s0	*s'[0]*	*s1*	*s'[1]*	*s2*	*s'[2]*	*s3*	*s'[3]*	*s4*	*s'[4]*	*s5*
d	*p*	*u*	*i*	*vv*	*c*	*j*	*k*	*reer*	*t*	
	p	*l*	*i*		*c*		*k*	*e*	*t*	*s*
t0	*s'[0]*	*t1*	*s'[1]*	*t2*	*s'[2]*	*t3*	*s'[3]*	*t4*	*s'[4]*	*t5*

Tabela 1. Alinhamento entre as strings dpuivvcjkreert e picket (caracteres nas listas de deleções exibidos em fundo cinza)

Este alinhamento revela quatro possíveis situações: (i) substring deve ser deletada (ex: *d* em s_0, *vv* em s_2), (ii) caracteres devem ser substituídos (ex: *u* por *l* em s_1); (iii) substring deve ser inserida (ex: *s* em s_5), e (iv) alinhamento mais complexo deve ser realizado (ex: *reer* → *e* em s_4).

O *AMPH-QHD* basicamente reconhece estas quatro situações, executando o algoritmo *de Damerau-Levenshtein* apenas para o caso (iv). Ou seja, para os casos (i), (ii) e (iii) são realizadas operações simples de deleção, substituição e inserção, respectivamente. Ao fim do processo, a lista é operações realizadas é reavaliada para verificar se pares de operações escolhidas não correspondem a transposições. Este é o caso, por exemplo, da deleção do caractere c_1 após o caractere c_2, seguido da inserção de c_1. Estas duas operações correspondem a uma transposição de c_1 por c_2. Se este é o caso, a transposição é colocada no lugar das operações avaliadas.

Como se observa no algoritmo 4, em *AMPH-QHD a*s operações são alinhadas em correspondência com o tamanho das *substring* (linhas 6-13), esse passo é muito importante porque possibilita o perfeito alinhamento das operações dos caracteres em *s* e *t*, colocando as mesmas nas posições correspondentes das listas de caracteres *Ls[1..n]* e *Lt[1...n]* onde *n* e o número de operações que feitas pelo Mor-Fraenkel**,** (linhas 15-30). Em esta união, o algoritmo 3 considera que em presencia de uma substituição complexa, ou seja quando existem muitas formas de transformar um residual de *s* em um residual de *t,* a operação de transposição aplica-se se aos residuais que são inversamente iguais, caso contrário a solução ótima é dada pelo método de Damerau-Levenshtein (linha 30).

```
1:  function AMPH-QHD(s, t, P^s, P^t)
2:      E ← []                          ▸ List of edit operations
3:      L_s ← []                        ▸ List of edit operations of s
4:      L_t ← []                        ▸ List of edit operations of t
5:                                      ▸ Alignment op in P^s and P^t
6:      for w ← 0 to BiggerSize(P^[s,t]) do
7:          if P^[s,t] is out then
8:              add(P^[s,t],null) to L_[s,t][m]
9:          else
10:             if P^[s,t] is match then
11:                 add(P^[s,t],match) to L_[s,t][m]
12:             else
13:                 add(P^[s,t],other) to L_[s,t][m]
14:                                     ▸ Join (L^t and L^s)
15:     for m ← 0 to Size(L_t) do L_t[m] U L_s[m]
16:         if match then
17:             E[m] ← MATCH( L_t[m],L_s[m])
18:         else if insertion then
19:             E[m] ← INS(L_s[m],L_t[m])
20:         else if deletion then
21:             E[m] ← DEL(L_s[m])
22:         else if substitution then
23:             if simple-substitution then
24:                 E[m] ← SUST(L_t[m],L_s[m])
25:             else if iqual-inverse then
26:                 E[m] ← TRAS(L_s[m])
27:             else
28:                 for i ← 1 to m do
29:                     for j ← 1 to n do
30:                         E[m] ← min d[0..m_s, 0..n_t]
                                d[i − 1, j] + 1,
                                d[i, j − 1] + 1,
                               d[i − 1, j − 1] + c
31:     return E
```

Figura 9. Algoritmo 4 (AMPH-QHD)

Isto possibilita uma diminuição na quantidade de operações a serem feitas e assim ganho no tempo de execução, devido que nestes casos Damerau-Levenshtein sempre vai fornecer a solução ótima. Nos demais casos são aproveitadas as operações que foram definidas pelo método de Mor-Fraenkel.

Em suma, *AMPH-QHD* trata casos mais complexos de alinhamento usando o algoritmo de Damerau-Levenshtein. O custo médio de *AMPH-QHD*, para um par de strings *s* e *t*, é inferior ao do método de Damerau-Levenshtein que é proporcional a *O(|s t|)*. No pior caso, contudo, é ainda quadrático. O pior caso ocorre quando é necessário alinhar substrings s_i e t_i com comprimentos similares às strings originais *s* e *t*, ou seja, se *s* e *t* casam no início ou fim das strings e têm longos resíduos.

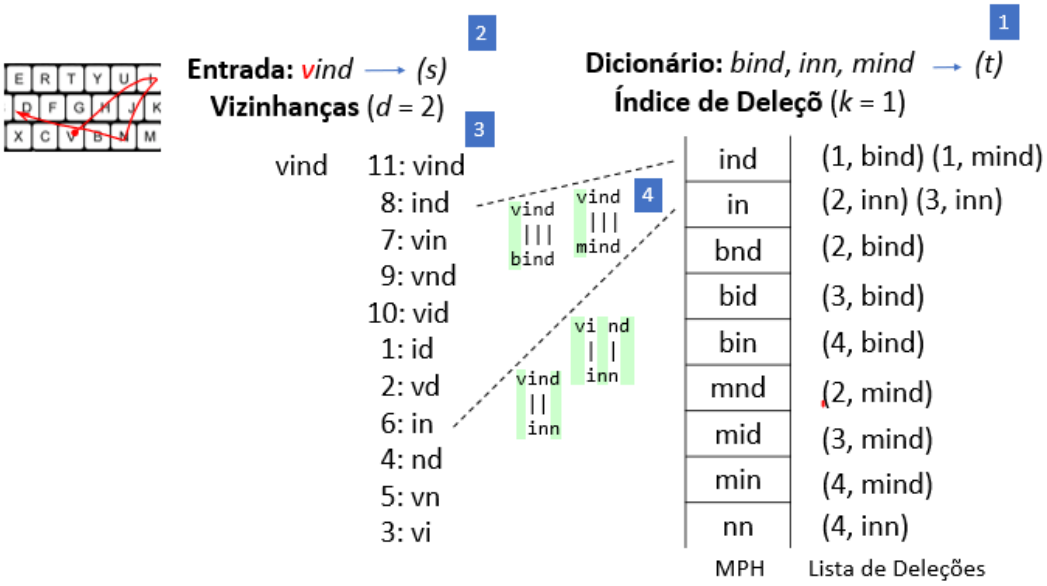


Figura 10. Abordagem usando Hashing Perfeito Mínimo

A Figura 10 ilustra o uso de hashing perfeito mínimo nos métodos propostos. Como se observa, no passo 1 nossa abordagem cria os índices de deleções usando um hashing perfeito mínimo em lugar da Trie usada em Hanada et al. (2016). No passo 2 o sistema recebe a palavra de entrada *s* que o usuário digitou, e no passo 3 cria e ordena as correspondentes vizinhanças de *s* de forma que os primeiros residuais a avaliar serão aqueles com menor número de deleções. Finalmente no passo 4 é utilizada a heurística linear de estimação de Mor-Fraenkel para alinhar os residuais de *s* e *t*. A partir desse alinhamento é possível identificar quais foram os casos de match, inserção, deleção, substituição e transposição que permitem

transformar estimar o custo de transformar os residuais na string correta. O uso de hashing perfeito mínimo nos permitiu ganhar uma maior rapidez no tempo de pesquisa .

Capitulo 5

EXPERIEMNTOS E RESULTADOS

5.1 Trabalhos de Referência

Com nossos métodos se concentram em cenários caracterizados pelo ruído, nossa principal linha de base é o trabalho relatado por Hanada et al. (2016), a que nos referimos como *CIKM16*. Como uma abordagem de dicionário aproximado rápido, adotamos o trabalho de Boytsov (2012), que denominamos *SISAP12*, que adota uma abordagem de Mor-Fraenkel com hash perfeito para buscar candidatos usando a distância de edição de Damerau-Levenshtein ≤ *k*. Em relação ao *SISAP12*, implementamos uma versão em Java com base em seu código original que estava em C++. Observamos que a implementação original de *SISAP12* comprime o índice de Mor-Fraenkel para avaliar seu impacto na escalabilidade. Como Boytsov usa uma técnica de compressão muito simples e aplicável a qualquer índice Mor-Fraenkel, também pode ser usado pelo *CIKM16* e nossos métodos. No entanto, como o *CIKM16* original não usa qualquer compressão de índice, para fornecer uma comparação justa, então desativamos a compressão no *SISAP12*.

No *SISAP12*, os candidatos são classificados de acordo com a distância e frequência de palavras, enquanto o *CIKM16* classifica os candidatos de acordo com suas probabilidades de edição específicas extraídas de um corpus de erro.

5.2 Metodologias de Avaliação

Utilizamos o conjunto de dados fornecido por Hanada et al. (2016) como nossa lista de palavras com ruído para o reconhecimento das mesmas. Este conjunto de dados inclui cadeias digitadas usando eye-tracker, onde foi aplicado um filtro para reduzir o ruído no sistema. Este conjunto de dados contém 3.073 pares de string *s* e *t*. De agora em diante nos referimos a este conjunto de dados como *EYE-PAIRS.*

Como nossos métodos usam a Eq.2 para calcular a semelhança entre os candidatos (*s* e *t*), precisávamos calcular L_i, L_d, L_s, L_t e L_m para *EYE-PAIRS*. Essas variáveis correspondem aos logaritmos negativos das probabilidades P_i, P_d, P_s, P_t e P_m. Para *EYE-PAIRS*, utilizamos as estimativas adotadas em Hanada et al. (2016). Escolhemos dividir em seis vezes o conjunto de dados *EYE-PAIRS*, porque contém pares gerados por seis usuários diferentes. Assim, dividimos os pares em 6 partições, onde cada partição corresponde a um usuário diferente. Como resultado, as estimativas de probabilidade e todas as métricas de avaliação relatadas para *CIKM16*, *AMPH-MML*, *AMPH-LHD* e *AMPH-QA* foram obtidas como média de seis rodadas: em cada rodada, uma partição diferente foi usada como conjunto de teste onde as partições restantes foram usadas para treinamento. As estimativas de probabilidade que obtivemos são mostradas na tabela 2.

Folds	#pairs	P_i	P_d	P_s	P_t	P_m
EYE-PAIRS						
1	534	0.043	0.219	0.083	0.001	0.653
2	558	0.018	0.243	0.034	0.001	0.703
3	590	0.022	0.259	0.064	0.001	0.654
4	531	0.014	0.275	0.057	0.001	0.652
5	478	0.033	0.272	0.061	0.001	0.632
6	382	0.007	0.468	0.053	0.001	0.471

Tabela 2. Probabilidades de operações de edição em EYE-PAIRS

Para estimar os logaritmos das probabilidades das operações específicas de edição, usamos frequências de operações de edição observadas no corpus geral usado por Norvig (2009), e as informações específicas de ruído fornecidas por Hanada et al. (2016). Para *EYEPAIRS*, como em Hanada et al. (2016) consideramos

L_E = - *log* (0.3P_C + 0.7P_N), onde P_C indicou probabilidades associadas ao corpus geral e P_N as probabilidades associadas ao ruído. Os conjuntos de dados com frequências de palavras (dicionários) e frequências de operações de edição podem ser baixados gratuitamente desde (*http://norvig.com/ngrams)*. As operações de edição basearam-se principalmente no erro de ortografia de Birkbeck, uma coleção com 36.133 erros ortográficos de 6.136 palavras, do Oxford Text Archive.

Adotamos um modelo de *unigram simples* com frequências de palavras extraídas de dicionários públicos. Este modelo pode ser facilmente estendido para suportar *n-grams*, $n > 1$, se o espaço adicional necessário para armazenar as estatísticas de n-grams não é uma preocupação.

Comparamos os métodos propostos de acordo com o tempo, espaço e qualidade de classificação, fornecendo intervalos de confiança e resultados de teste de significância, para um nível de confiança de 95%, quando necessário.

Avaliamos os rankings gerados pelos algoritmos usando a medida estatística *MRR,* que corresponde ao ranking reciproco médio (Voorhees,1999). *MRR* é a média do inverso das classificações em que a palavra correta foi recuperada para um conjunto de padrões de busca. Assim, dada uma entrada s_i, se a palavra correta *w* foi classificada na posição *r* pelo método *M, M é* classificado com ranking reciproco *RRi = 1/r.* Se a palavra correta não faz parte da lista de sugestões fornecidas por *M*, *M* é pontuado com 0. Esta métrica dá mais valor para as palavras no topo da lista de sugestões, sendo mais apropriada para problemas onde não existe muito espaço para mostrar sugestões.

Quanto à eficiência espacial, os métodos foram avaliados de acordo com o tamanho de seus índices em bytes, após descontar a sobrecarga de meta-dados em Java. Testamos os métodos com amostras de 70k, 140k e 280k palavras extraídas do dicionário principal fornecido por Hanada et al. (2016). Tomamos medidas como em Boytsov (2011) para a eficiência do tempo, ou seja, cada experimento foi realizado duas vezes, e somente o tempo menor foi gravado para minimizar interferências de processos do SO. Todos os experimentos foram realizados em uma máquina Intel i7 de 6 núcleos de 3,6 Ghz com 32 GB de RAM, Linux OS kernel 4.4.x e JVM HotSpot 64-Bit Server.

5.3 Resultados

Começamos definindo com décima nona letra do alfabeto grego *tau* um parâmetro de tempo limite apropriado τ para o processamento de poda e melhores valores de *d* em *AMPH-MML*, *AMPH-LHD* e *AMPH-QA*. Em todos os experimentos relatados aqui, se não for mencionado de forma diferente, usamos *k* = 2, pois com esse valor foram obtidos os melhores resultados em nossos experimentos preliminares, o que confirmou os resultados de Hanada et al. (2016).

Foram processadas as strings residuais de nosso conjunto de dados *EYE-PAIRS* de duas formas: ordenada de acordo com o número de deleções (sorted*),* e de forma aleatória natural *(*random). A Figura 11 mostra o efeito do processamento das strings para ambos casos, de acordo com o número de deleções suportadas pelo *AMPH-QHD*, usando dicionário de 70k e τ = 64 ms. Como se observa, a ordem aleatória levou ao pior desempenho, especialmente para um alto valor de *d*. Com mais deleções, podem ser obtidos mais resíduos, aumentando o número de candidatos a avaliar. Assim, a possibilidade que tem a palavra correta de estar no grupo em que foi avaliada antes do tempo limite é menos provável. Como resultado, o desempenho da abordagem aleatória foi cada vez pior.

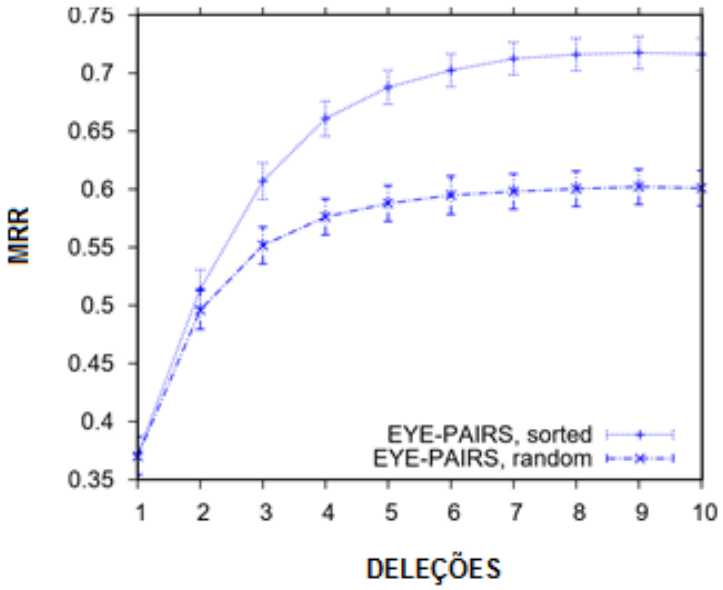


Figura 11. Processamento das strings usando o dicionário 70k e τ = 64 ms

Consideramos os três tamanhos de dicionário, e o conjunto de dados *EYE-PAIRS*, foram definidos para nossos experimentos uma quantidade máxima de operações de edição de caracteres *k*, e de deleção de caracteres *d*. Comparamos nossos métodos com os trabalhos base *SISAP12* (*k* = 1 a 4) e CIKM16 (*k* = 1 a 3, *d* = 10). A figura 12 mostra o tamanho do índice em relação ao MRR. Nessa figura, para cada método, existem três valores MRR correspondentes ao uso dos dicionários com 70k, 140k e 280k palavras, respectivamente.

Primeiro observamos que o índice *CIKM16* é maior do que os outros métodos, devido ao uso de estruturas Trie em vez de hashing perfeito. Ficou claro que a métrica baseada em MML usada por *CIKM16*, *AMPH-MML*, *AMPH-LHD*, e *AMPH-QHD* valem a pena, resultando em ganhos consistentes em relação ao *SISAP12*.

CIKM16, *AMPH-MML* e *AMPH-QHD* apresentaram níveis equivalentes de MRR, enquanto a *AMPH-LHD* foi ligeiramente inferior.

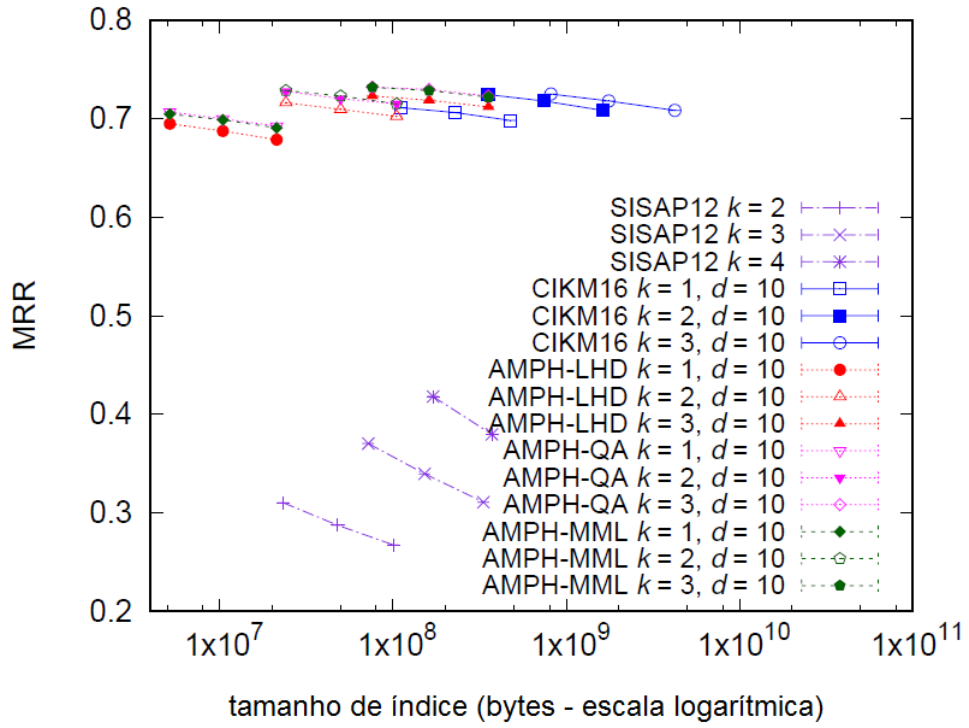


Figura 12. Tamanho de índice em relação ao MRR

Em quanto eficácia no tempo, a figura 13 mostra o tamanho do índice em relação ao tempo médio de resposta. Como antes, para cada método havia três valores correspondentes aos dicionários de 70k, 140k e 280k palavras respectivamente. Pode se observar que *AMPH-LHD*, *AMPH-MML* e *AMPH-QHD* presentaram um desempenho muito parecido. Isto se deve a que na maioria dos casos ele tem complexidade parecidas. Por em nós conseguimos uma melhoria em comparação ao tempo de processamento dos métodos *SISAP12* e *CIKM16*. A diferencia, mas notável no tempo de processamento foi aproximadamente de 50 a 170 vezes mais rápido que o método *CIKM16*, dependendo do tamanho do dicionário e de *k*. Considerando o MRR elevado obtido pelos métodos *AMPH-LHD*, *AMPH-MML* e *AMPH-QA*, pode-se afirmar que os mesmos foram mais eficazes em reconhecimento de palavras do que *SISAP12*, e mais rápidos do que o *CIKM16*, com pouca ou nenhuma perda de qualidade de reconhecimento.

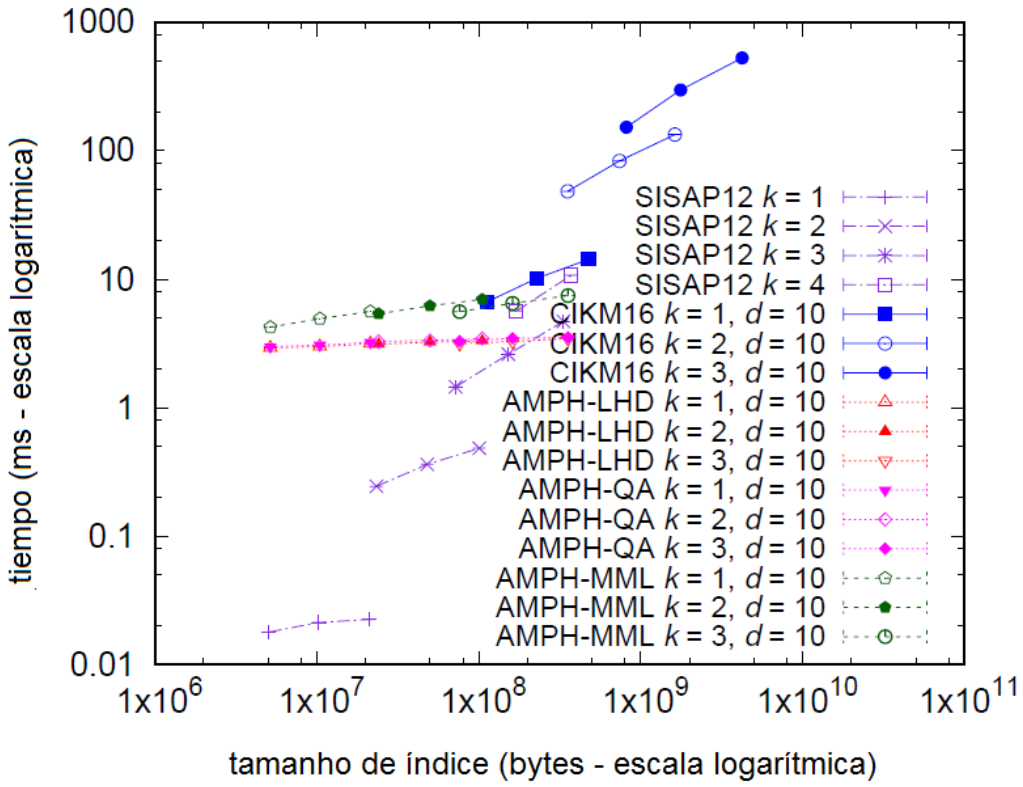


Figura 13. Tamanho de índice em relação ao tempo

Capítulo 6

CONCLUSÕES

Neste trabalho, apresentamos 3 métodos para reconhecer palavras em cenários com muito ruído. *AMPH-MML* recupera os candidatos ao combinar um índice assimétrico Mor-Fraenkel com um hash mínimo perfeito. Em seguida, classifica os candidatos usando uma métrica de distância baseada em *MML*. *AMPH-LHD* estende o *AMPH-MML* substituindo a métrica *MML* por uma heurística linear que se beneficia da ordem de processamento dos resíduos e informações armazenadas no índice Mor-Fraenkel. *AMPH-QA* trata casos mais complexos*,* ele utiliza toda a informação que brinda Mor-Fraenkel para diminuir a quantidade de operações a serem feitas, usando o alinhamento ótimo de Damerau-Levenshtein.

Em particular, o processamento ordenado possibilita que os candidatos mais promissores sejam avaliados com antecedência, permitindo estratégias de podas baseadas em limitar o tempo de processamento. Ao testar nossos métodos usando conjuntos de dados de reconhecimento de palavras e correção ortográfica, mostramos que os métodos propostos são muito eficazes. Eles são na maior parte superior aos trabalhos de base que foram propostos, em relação à eficiência do tempo, sendo 25 a 170 vezes mais rápido, dependendo do tamanho do dicionário e do dicionário. Quanto à precisão do reconhecimento de palavras em cenários ruidosos, *AMPHMML* e *AMPH-QA* alcançaram a mesma precisão do *CIKM16* enquanto o *AMPH-LHD* foi apenas um pouco menor.

Um problema que existe com nossos algoritmos é a dificuldade de atualização do dicionário, dado o alto custo de construção de um hash mínimo perfeito. No futuro, pretendemos melhorar isso usando hash hierárquicas mínimas perfeitas, ou outras estratégias eficientes, como a proposta Chegrane et al. (2013). Este trabalho,

em particular, é tão rápido e eficiente em termos de espaço quanto os nossos métodos, mas com um tempo de construção mais rápido. Para usá-lo, contudo, precisamos muda-lo para que incorpore transposições.

REFERÊNCIAS

Baba, Y., Suzuki, H. (2012). How are spelling errors generated and corrected: a study of corrected and uncorrected spelling errors using keystroke logs. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2 (ACL '12), Vol. 2. Association for Computational Linguistics, Stroudsburg, PA, USA. p 373-377.

Barreto, M. (2012). Eye tracking como método de investigação aplicado às ciências da comunicação. Revista Comunicando, 1(1). p 168-186.

Beers, V., Robert, J. (2007). The sources of variability in saccadic eye movements. The Journal of Neuroscience, 27(33). p 8757-8770.

Beers, V., Robert, J. (2010). Correction: saccadic eye movements minimize the consequences of motor noise. PLoS one, 5(7).

Blanchard, E., Pollatsek, A., Rayner, K. (1989). The acquisition of parafoveal word information in reading. Percept. Psychophys. p 85-94.

Brookings, J. B., Wilson, G. F., Swain, C. R. (1996). Psychophysiological responses to changes in workload during simulated air traffic control. Biological Psychology. p 361-377.

Boytsov, L. (2011). Indexing methods for approximate dictionary searching: Comparative analysis. Journal of Experimental Algorithmics (JEA), https://doi.org/10.1145/1963190.1963191. Article 1.1.

Boytsov, L. (2012). Super-linear indices for approximate dictionary searching. In International Conference on Similarity Search and Applications. Springer Berlin Heidelberg. p 162-176.

T, Bocek., E., Hunt., B, Stiller. (2007). Fast Similarity Search in Large Dictionaries.Technical Report ifi-2007.02. U. Zurich. http://fastss.csg.uzh.ch/.

Botelho, F., Pagh, R., Ziviani, N. (2007). Simple and Space-Efficient Minimal Perfect Hash Functions. Springer, Berlin, Heidelberg. DOI:http://dx.doi.org/10.1007/978-3-540-73951-7_13. p 139-150.

Brill, E., R. Moore. (2000). An improved error model for noisy channel spelling correction. In Proceedings of the ACL 2000. p 286-293.

Cristo, M., Hanada, R., Carvalho, A., Anglada, F., Pimentel, M. (2017). FastWord Recognition for Noise channel-based Models in Scenarios with Noise Specific Domain Knowledge. CIKM'17, Singapore, p 607-616.

Choil, K. (2014). Improving the Usability of Remote Eye Gaze Tracking for HumanDevice Interaction. Vision Research. p 929-942.

Church, K., Gale, W. (1991). Probability Scoring for Spelling Correction. Statistics and Computing. http://publication.wilsonwong.me/load.php?id=233281639. p 93–103.

Chegrane, I., Belazzougui, D. (2013). Simple, compact and robust approximate string dictionary. CoRR abs/1312.4678 (2013). http://dblp.uni-trier.de/db/journals/corr/corr1312.html#ChegraneB13

Damerau, F. (1964). A technique for computer detection and correction of spelling errors. CACM 7. DOI:http://dx.doi.org/10.1145/363958.363994. p 171-176.

Deubel, H., Schneider, W. (1996). Saccade target selection and object recognition: Evidence for a common attentional mechanism. Vision Research 36, p 1827-1837.

Demer, J., Miller, M., Poukens, V., Vinters, H., Glasgow, B. (1995) Evidence for fibromuscular pulleys of the recti extraocular muscles. Invest Ophthalmol Vis Sci 36. p 1125–1136.

Fredkin, E. (1960). Trie Memory. ACM 3. DOI:10.1145/367390.367400. p 490-499.

Enderle, A., Zhou, H. (2010). Models of Horizontal Eye Movements, Part II: A 3rd Order Linear Saccade Model. Morgan & Claypool Publishers.

Enderle, J. (2000). The Fast Eye Movement Control System. The Biomedical Engineering Handbook: Second Edition. p 166-200.

Findlay, J., Walker, R. (2012). Human saccadic eye movements. *Scholarpedia*, *7*(7). p 5095.

Gorin, R. (1971). Spelling check and correction program. http://pdp-10.trailing-edge.com/decuslib10-03/01/43,50270/spell.doc.html. Accessed 10 Abr 2016.

Goldberg, J., Wichansky, A. (2003). Eye tracking in usability evaluation: A Practitioner's Guide. In: Hyona, J., Radach, R., Duebel, H (Eds.). The mind's eye: cognitive and applied aspects of eye movement research. Boston, North-Holland / Elesevier. p 573-605.

Garbe, W. (2012). Improved Edit-distance based Spelling Correction. Faroo Search Engine. Available in web at http://blog.faroo.com/2012/06/07/improved-edit-distancebased-spelling-correction/.

Hanada, R., Pimentel, M., Cristo, M., Anglada, F. (2016). Effective Spelling Correction for Eye-based Typing Using Domainspecific Information About Error Distribution (CIKM'16). ACM, New York, NY, USA, tps://doi.org/10.1145/2983323.2983838. p1723–1732.

Harwood, A., Herman, M. (2008). Optimally Straight and Optimally Curved Saccades. The Journal of Neuroscience 28, 30. p 7455-7457.

Harris, C., Wolpert, D. (1998). Signal-dependent noise determines motor planning. Nature 394. p 780–784.

Harris, C. (1998); On the optimal control of behaviour: a stochastic perspective. J Neurosci Meth 83. p 73–88.

Haque, W., Aravind, A., Reddy, B. (2009). Pairwise sequence alignment algorithms: a survey. In Proceedings of the 2009 conference on Information Science, Technology and Applications (ISTA '09). p 96-103.

Hoppe, S., Löchtefeld, M., Daiber, F. (2013). Eype—Using eye-traces for eye-typing. In Workshop on Grand Challenges in Text Entry.

Hocking, L. (1991). Optimal Control. An Introduction to the Theory with Applications. Oxford: Clarendon Press.

Jayarathna, S., (2010). Thesis: Two Dimentional Oculomotor Plant Mechanical Model. M.S Computer Science. p 1-64.

Jarodzka, H., Holmqvist, K., Nystr, M. (2010). A vector-based, multidimensional scanpath similarity measure. In Proceedings of the Symposium on Eye-Tracking Research Applications, Austin, Texas. p 211-218.

Just, M., Carpenter, P. (1980). A theory of reading: from eye fixation to comprehension. Psychol 87. p 329–354.

Javal, L. É. (1878). Essai sur la physiologie de la lecture. In Annales d'Oculistique. 79. p 97-117.

Jurafsky, D., Martin, J. H. (2000). Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.

Kernighan, M., Church, K., Gale, W. (1990). A Spelling Correction Program Based on a Noisy Channel Model. COLING'90.
DOI:http://dx.doi.org/10.3115/997939.997975. p 205–210.

Kristensson, P., Vertaneny, K. (2012). Department of Computer Science Montana Tech of the Uniersity of Montana.

Komogortsev, O., Holland, C., Jayarathna, S., Karpov A., (2014). 2D Linear Oculomotor Plant. Mathematical Model: Verification and Biometric Applications". ACM Transactions on Applied Perception (TAP), Vol. 10(4).

Komogortsev, O., Khan, J. (2009). Eye Movement Prediction by Oculomotor Plant Kalman Filter with Brainstem Control. Journal of Control Theory and Applications. Vol 7. Issue 1. p 386-493.
Komogortsev, K., Khan, D. (2009). Instantaneous Saccade Driven Eye Gaze Interaction.

Kirk, D. (1970). Optimal Control Theory. An Introduction. Englewood Cliffs, New Jersey: Prentice-Hall.

Komogortsev, O., Khan, J. (2007). Kalman Filtering in the Design of Eye-GazeGuided Computer Interfaces [C]. Proceedings of the 2007 12th International Conference on Human-Computer Interaction.

Komogortsev, O., Khan, J. (2008). Eye Movement Prediction by Kalman Filter with Integrated Linear Horizontal Oculomotor Plant Mechanical Model. Proceedings of the 2008 ACM Symposium on Eye Tracking Research & Applications, p 229-236.

Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions and reversals. Sov. Phys. Dokl. p 707-710.

Leigh RJ, Zee DS. (2006). The Neurology of Eye Movements (Book/DVD). 4ed. New York: Oxford University Press.

Li, H., Homer, N. (2010). A survey of sequence alignment algorithms for nextgeneration sequencing Briefings in Bioinformatics, Vol. 11, No. 5. September 2010. p 473-483.

Lathi, B. (1965). Signals, systems and communication. New York: John Wiley & Sons, ISBN.

Liechty, J., Pieters, R., Wedel, M. (2003). The Representation of Local and Global Exploration Modes in Eye Movements through Bayesian Hidden Markov Models. Psychometrika. p 519–542.

A, Lloyd., Wallace, C., Yee, N. (1990). Minimum message length encoding and the comparison of macromolecules. Bulletin of Mathematical Biology 52. DOI:http://dx.doi.org/10.1007/bf02458580. p 431-453.

Mays, E., Damerau, F., Mercer, R. (1991). Context Based SpellingCorrection. Information Processing & Management 27. DOI:http://dx.doi.org/10.1016/0306-4573(91)90066-U. p 517-522.

M, Mor., S, Fraenkel. (1982). A Hash Code Method for Detecting and Correcting Spelling Errors. CACM 25, https://doi.org/10.1145/358728.358752, p 935–938.

Morimoto, C., Amir, A. (2010). Context switching for fast key selection in text entry applications. In Proceedings of the 2010 Symposium on Eye-Tracking Research. p 271.

Mannan, S.K., Ruddock, K., Wooding, D. (1996). The relationship between the locations of spatial features and those of fixations made during visual examination of briefly presented images. Spatial. Vision, 10(3). p 165-188.

Mannan, S.K., Ruddock, K., Wooding, D. (1997). Fixation sequences made during visual examination of briefly presented 2D images. Spatial Vision, 11(2). p 157-178.

Meur, O., Baccino, T. (2012). Methods for comparing scanpaths and saliency maps: strengths and weaknesses. HAL. <hal-00757615>.

Carvalho, J. (2012). O papel da interação humano-computador na inclusão digital. Transinformação, 15(3).

Norvig, P. (2009). Natural Language Corpus Data. In Beautiful Data: The Stories Behind Elegant Data Solutions, Toby Segaran and Jeff Hammerbacher (Eds.). O'Reilly Media. http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0596157118.

Noton, D., Stark L. (1970). Eye movements and visual perception. Scientific American. p 929-942.

Noton, D., Stark L. (1971). Scanpaths in saccadic eye movements while viewing and pattern recognition. Vision Research. p 929-942.

Rayner, K. (1978). Eye movements in reading and information processing. Psychological Bulletin 85. p 618-660.

Rayner, K. (1998). Eye Movements in Reading and Information Processing: 20 Years of Research. Psychological Bulletin. p 372-422.

Shannon, C. (1948). A mathematical theory of communication. Bell System Technical Journal 27(3). p 379-423.

Shaun, K., Wobbrock, J., Harniss, M., Kurt, L. (2008). TrueKeys: Identifying and Correcting Typing Errors for People with Motor Impairments. In ACM IUI '08. https://doi.org/10.1145/1378773.1378827. p 349-352.

Pedrosa, D., Pimentel, M. D. G., Wright, A., Truong, K. N. (2015). Filteryedping: Design challenges and user performance of dwell-free eye typing. ACM Transactions on Accessible Computing.

POMPLUN, M., SUNKARA, S. (2003). Pupil dilation as an indicator of cognitive workload in HumanComputer Interaction. In: Proceedings of HCI International 2003: Mahwah, NJ: Lawrence Erlbaum Associates. Vol 3. p 542-546.

Pedrosa, D., Pimentel, M., Wright, M., Truong K. (2015). Filteryedping: Design Challenges and User Performance of Dwell-Free Eye Typing. ACM Trans. Access. Comput. 6, 1. https://doi.org/10.1145/2724728, Article 3.

Norvig, P. (2009). Natural language corpus data. Beautiful data: the stories behind elegant data solutions. p 219-242.

Opstal, AJ., Gisbergen, J. (1989) Scatter in the metrics of saccades and properties of the collicular motor map. Vision Res 45. p 1183-1196.

Robinson, D. (1964). The mechanics of human saccadic eye movement. J Physiol 174. p 245-264.

Robinson, D., Gordon, J. (1986). A model of the smooth pursuit eye movement system. Biol Cybern. p 43-57.

Raphan, T. (1998). Modeling the control of eye orientation in three dimensions. I. Role of muscle pulleys in determining saccadic trajectory. J Neurophysiol 79. p 2653– 667.

Schutte, S., Bedem, V., Keulen, F., Simonsz, H. (2006). A finite element analysis model of orbital biomechanics. Vision Res 46. p 1724-1731.

Schnabolk, C., Raphan, T. (1994). Modeling three-dimensional velocity-to-position transformation in oculomotor control. J Neurophysiol 71. p 623-638.

Tatler, B. W., Wade, N. (2003). On nystagmus, saccades, and fixations. *Perception*, *32*(2). p 167-184.

Tweed, D., Vilis, T. (1987). Implications of rotational kinematics for the oculomotor system in three dimensions. J Neurophysiol 58. p 823–849.

Tamae, R., Pimentel, M. (2015). Improving Dwell-Free Eye typing techniques using a Deep Learning approach. Research Proposal for a 12-month internship at the University of Toronto (UoT), Canada.

Voorhees, E. (1999). The TREC-8 Question Answering Track Report. In Proc. TREC'1999. http://trec.nist.gov/pubs/trec8/papers/qa_report.pdf

Zhai, S., Kristensson, P. (2012). The word-gesture keyboard: reimagining keyboard interaction. Communications of the ACM, 55(9). p 91-101.