UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

A noisy-channel based model to recognize words in eye typing systems

Raíza Tamae Sarkis Hanada

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Raíza Tamae Sarkis Hanada

A noisy-channel based model to recognize words in eye typing systems

Doctoral dissertation submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Profa. Dra. Maria da Graça Campos Pimentel

USP – São Carlos
February 2018

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

S233n Sarkis Hanada, Raiza Tamae
 A Noisy-channel based Model to Recognize Words
 in Eye Typing Systems / Raiza Tamae Sarkis Hanada;
 orientadora Maria da Graça Campos Pimentel. -- São
 Carlos, 2018.
 110 p.

 Tese (Doutorado - Programa de Pós-Graduação em
 Ciências de Computação e Matemática Computacional) --
 Instituto de Ciências Matemáticas e de Computação,
 Universidade de São Paulo, 2018.

 1. Eye-based Typing Systems. 2. Noisy-Channel
 Models. 3. Human-computer Interfaces. 4. Mor-
 Fraenkel Indexes. I. Pimentel, Maria da Graça
 Campos, orient. II. Título.

Raíza Tamae Sarkis Hanada

**Um modelo baseado em canal de ruído para reconhecer
palavras digitadas com os olhos**

Tese apresentada ao Instituto de Ciências
Matemáticas e de Computação – ICMC-USP,
como parte dos requisitos para obtenção do título
de Doutora em Ciências – Ciências de Computação
e Matemática Computacional. *EXEMPLAR DE
DEFESA*

Área de Concentração: Ciências de Computação e
Matemática Computacional

Orientadora: Profa. Dra. Maria da Graça
Campos Pimentel

USP – São Carlos
Fevereiro de 2018

ABSTRACT

HANADA, R. T. S.. **A noisy-channel based model to recognize words in eye typing systems.** 2018. 110 f. Doctoral dissertation (Doctorate Candidate Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

An important issue with eye-based typing is the correct identification of both when the user selects a key and which key is selected. Traditional solutions are based on predefined gaze fixation time, known as dwell-time methods. In an attempt to improve accuracy long dwell times are adopted, which in turn lead to fatigue and longer response times. These problems motivate the proposal of methods free of dwell-time, or with very short ones, which rely on more robust recognition techniques to reduce the uncertainty about users' actions. These techniques are specially important when the users have disabilities which affect their eye movements or use inexpensive eye trackers. An approach to deal with the recognition problem is to treat it as a spelling correction task. An usual strategy for spelling correction is to model the problem as the transmission of a word through a noisy-channel, such that it is necessary to determine which known word of a lexicon is the received string. A feasible application of this method requires the reduction of the set of candidate words by choosing only the ones that can be transformed into the input string by applying up to k character edit operations. This idea works well on traditional typing because the number of errors per word is very small. However, this is not the case for eye-based typing systems, which are much noisier. In such a scenario, spelling correction strategies do not scale well as they grow exponentially with k and the lexicon size. Moreover, the error distribution in eye typing is different, with much more insertion errors due to specific sources, of noise such as the eye tracker device, particular user behaviors, and intrinsic characteristics of eye movements. Also, the lack of a large corpus of errors makes it hard to adopt probabilistic approaches based on information extracted from real world data. To address all these problems, we propose an effective recognition approach by combining estimates extracted from general error corpora with domain-specific knowledge about eye-based input. The technique is able to calculate edit distances effectively by using a Mor-Fraenkel index, searchable using a minimum perfect hashing. The method allows the early processing of most promising candidates, such that fast pruned searches present negligible loss in word ranking quality. We also propose a linear heuristic for estimating edit-based distances which take advantage of information already provided by the index. Finally, we extend our recognition model to include the variability of the eye movements as source of errors, provide a comprehensive study about the importance of the noise model when combined with a language model and determine how it affects the user behaviour while she is typing. As result, we obtain a method very effective on the task of recognizing words and fast enough to be use in real eye typing systems. In a transcription experiment with 8 users, they achieved 17.46 words per minute using the proposed model, a

gain of 11.3% over a state-of-the-art eye-typing system. The method was particularly useful in more noisier situations, such as the first use sessions. Despite significant gains in typing speed and word recognition ability, we were not able to find statistically significant differences on the participants' perception about their experience with both methods. This indicates that an improved suggestion ranking may not be clearly perceptible by the users even when it enhances their performance.

Key-words: Eye-based Typing Systems, Noisy-Channel Models, Human-computer Interfaces, Mor-Fraenkel Indexes.

RESUMO

HANADA, R. T. S. **Um modelo baseado em canal de ruído para reconhecer palavras digitadas com os olhos**. 2018. 110 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2018.

Um problema importante em sistemas de digitação com os olhos é a correta identificação tanto de quando uma letra é selecionada como de qual letra foi selecionada pelo usuário. As soluções tradicionais para este problema são baseadas na verificação de quanto tempo o olho permanece retido em um alvo. Se ele fica por um certo limite de tempo, a seleção é reconhecida. Métodos que usam esta ideia são conhecidos como *baseados em tempo de retenção (dwell time)*. É comum que tais métodos, com intuito de melhorar a precisão, adotem tempos de retenção alto. Isso, por outro lado, leva à fadiga e tempos de resposta altos. Estes problemas motivaram a proposta de métodos não baseados em tempos de retenção ou com tempos de retenção reduzidos, que dependem de técnicas mais robustas de reconhecimento para inferir as ações dos usuários. Tais estratégias são particularmente mais importantes quando o usuário tem desabilidades que afetam o movimento dos olhos ou usam dispositivos de rastreamento ocular (*eye-trackers*) muito baratos e, portanto, imprecisos. Uma forma de lidar com o problema de reconhecimento das ações dos usuários é tratá-lo como correção ortográfica. Métodos comuns para correção ortográfica consistem em modelá-lo como a transmissão de uma palavra através de um canal de ruído, tal que é necessário determinar que palavra de um dicionário corresponde à string recebida. Para que a aplicação deste método seja viável, o conjunto de palavras candidatas é reduzido somente àquelas que podem ser transformadas na string de entrada pela aplicação de até k operações de edição de carácter. Esta ideia funciona bem em digitação tradicional porque o número de erros por palavra é muito pequeno. Contudo, este não é o caso de digitação com os olhos, onde há muito mais ruído. Em tal cenário, técnicas de correção de erros ortográficos não escalam pois seu custo cresce exponencialmente com k e o tamanho do dicionário. Além disso, a distribuição de erros neste cenário é diferente, com muito mais inserções incorretas devido a fontes específicas de ruído como o dispositivo de rastreamento ocular, certos comportamentos dos usuários e características intrínsecas dos movimentos dos olhos. O uso de técnicas probabilísticas baseadas na análise de logs de digitação também não é uma alternativa uma vez que não há corpora de dados grande o suficiente para tanto. Para lidar com todos estes problemas, propomos um método efetivo de reconhecimento que combina estimativas extraídas de corpus de erros gerais com conhecimento específico sobre fontes de erro encontradas em sistemas de digitação com os olhos. Nossa técnica é capaz de calcular distâncias de edição eficazmente usando um índice de Mor-Fraenkel em que buscas são feitas com auxílio de um hashing perfeito mínimo. O método possibilita o processamento ordenado de candidatos promissores, de forma que as operações de busca podem ser podadas sem que apresentem perda significativa na qualidade do ranking.

Nós também propomos uma heurística linear para estimar distância de edição, que tira proveito das informações já mantidas no índice, estendemos nosso modelo de reconhecimento para incluir erros vinculados à variabilidade decorrente dos movimentos oculares e fornecemos um estudo detalhado sobre a importância relativa dos modelos de ruído e de linguagem. Por fim, determinamos os efeitos do modelo no comportamento do usuário enquanto ele digita. Como resultado, obtivemos um método de reconhecimento muito eficaz e rápido o suficiente para ser usado em um sistema real. Em uma tarefa de transcrição com 8 usuários, eles alcançaram velocidade e 17,46 palavras por minuto usando o nosso modelo, o que corresponde a um ganho de 11,3% sobre um método do estado da arte. Nosso método se mostrou mais particularmente útil em situação onde há mais ruído, tal como a primeira sessão de uso. Apesar dos ganhos claros de velocidade de digitação, não encontramos diferenças estatisticamente significativas na percepção dos usuários sobre sua experiência com os dois métodos. Isto indica que uma melhoria no ranking de sugestões pode não ser claramente perceptível pelos usuários mesmo quando ela afeta positivamente os seus desempenhos.

Palavras-chave: Digitação com os Olhos, Modelos baseados em Canal Ruidoso, Interface Humano-Computador, Índices de Mor-Fraenkel.

LIST OF FIGURES

Figure 1 – Error distribution for $0 \leq k \leq 10$.	23
Figure 2 – Trie for keys I, in, inn, tea, ten, and to.	35
Figure 3 – Probability that point (x,y) belongs to key d in QWERTY keyboard scenario where dominant error source is the eye tracker with error radius of about 1° ($2 \sigma^2 = 1^\circ$).	45
Figure 4 – α x MRR for AMMLSEP, $k = 2$, $d = 10$.	49
Figure 5 – d x MRR and time, for AMMLSEP, $k = 2$, $\alpha = 0.7$.	49
Figure 6 – Space x MRR (AMMLSEP $\alpha = 0.7$).	50
Figure 7 – Space x time (AMMLSEP $\alpha = 0.7$).	50
Figure 8 – Size x MRR in 140k dictionary.	51
Figure 9 – Size x time (AMMLSEP $\alpha = 0.7$, $k = 2$, $d = 10$).	51
Figure 10 – Timeout τ versus MRR and average response time for AMPH-LHD.	65
Figure 11 – Input size versus response processing time.	66
Figure 12 – Deletions versus MRR for AMPH-LHD with processing of ordered and not ordered (random) string residuals in 70k dictionary.	66
Figure 13 – Index size versus MRR (EYE-PAIRS).	67
Figure 14 – Index size versus MRR (ASWIKI-PAIRS).	67
Figure 15 – Index size versus time (EYE-PAIRS).	68
Figure 16 – Index size versus time (ASWIKI-PAIRS).	68
Figure 17 – Error ellipsis resulting from sensory noise observed in saccade.	72
Figure 18 – Saccade endpoint probabilities	74
Figure 19 – Maximum pruning time (τ) x MRR for AMPH-LHD and C-AMPH-LHD.	79
Figure 20 – Effect of outlier and duplicate letter detection parameters on MRR.	79
Figure 21 – Best alpha for methods using or not contextual probabilities.	80
Figure 22 – MRR x importance of the language model (λ).	80
Figure 23 – Best alpha for proposed methods and CIKM'17 AMPH-LHD.	81
Figure 24 – Environment setup	85
Figure 25 – Filtered typing interface	86
Figure 26 – WPM per session.	90
Figure 27 – WPM per participants.	90
Figure 28 – WPM per participants for Sessions 1 and 6.	91
Figure 29 – Deletions per session.	91
Figure 30 – NASA TLX scores for Sessions 1 and 6.	93

Figure 31 – Final scores per scale for each condition. Me, Ph, Te, Pe Ef and Fr stand for Mental demand, Physical demand, Temporal demand, Perform, Effort and Frustration scales respectively. 93

Figure 32 – Percentage of pages visited by the participants when the correct word was not found. 97

Figure 33 – Percentage of words participants did not gazed to all letters per session. . . . 97

LIST OF TABLES

Table 1 – Character stream after being <i>detected</i> by an eye tracker and <i>filtered</i> out to remove noise	21
Table 2 – Error distribution in three different input systems	23
Table 3 – Notation and symbols used in this thesis	31
Table 4 – Methods used in this thesis to recognize words	40
Table 5 – Edit operation probabilities estimated from word pairs typed by six participants. Confidence intervals (Cf. Int.) are given for a 95% confidence level.	48
Table 6 – Proportion of the 3,073 pairs in which correct word was placed at top position and rank slots 1-6, 7-12, and 13-18.	52
Table 7 – Edit operation probabilities	63
Table 8 – Deletions versus MRR and average response time for AMPH-MML (confidence intervals given for 95% confidence level).	66
Table 9 – Proportion of the 3,073 pairs in which correct word was placed at top-1, 6, 12, and 18 positions.	81
Table 10 – General statistics for Filteredping and C-AMMLSEP-LHD.	88
Table 11 – Proportion of words found in Top-1, 6, 12, 18 positions. “All” stands for all suggestions found.	92
Table 12 – Time spent in each action per word (%).	94
Table 13 – Time spent in each action in average per word (ms).	95
Table 14 – Pages visited by participants until selecting a word.	96
Table 15 – Examples of suggestion lists associated with typed words which were not found.	97

LIST OF ABBREVIATIONS AND ACRONYMS

ALS	Amyotrophic Lateral Sclerosis
AMMLSEP	Asymmetrical Mor-Fraenkel with MML alignment and Specific Edit Probabilities
AMPH-LHD	Asymmetrical Mor-Fraenkel with MPH and a Linear Heuristic Distance metric
AMPH-MML	Asymmetrical Mor-Fraenkel with MPH and MML alignment
C-AMPH-LHD	Contextual Asymmetrical Mor-Fraenkel with MPH and a Linear Heuristic Distance metric
CIKM	ACM Conferences on Information and Knowledge Management
DGP	Dynamic Graphics Project Laboratory
DL	Damerau-Levenshtein
DMD	Duchenne Muscular Dystrophy
FAPEAM	Amazon Research Foundation
ICMC	Institute of Mathematics and Computer Sciences
IFX	Incorrect Fixed Rate
INFX	Incorrect not Fixed Rate
LHD	Linear Heuristic Distance
MF	Mor-Fraenkel
MML	Minimum Message Length
MPH	Minimum Perfect Hashing
MRR	Mean Reciprocal Rank
MSD	Minimum String Distance
NASA-TLX	NASA Task Load Index
OER	Overall Error Rate
SIGDOC	ACM International Conference on the Design of Communication
TOCHI	ACM Transactions on Computer-Human Interfaces
USP	University of São Paulo
WPM	Words per Minute

CONTENTS

1	INTRODUCTION	19
1.1	Context	19
1.2	Motivation	20
1.3	Problem Definition	21
1.4	Research Hypothesis and Questions	22
1.5	Objectives	25
1.6	Contributions	25
1.7	Thesis Organization	27
2	BACKGROUND AND RELATED WORK	29
2.1	Errors associated with eye-typing systems	29
2.2	Preliminary Notation	31
2.3	Noisy channel theory	32
2.4	Candidate ranking	32
2.4.1	<i>Classic edit distances</i>	32
2.4.2	<i>Related distance metrics</i>	34
2.5	Approximated dictionary search	35
2.5.1	<i>Related approximate searching methods</i>	36
2.6	Eye-typing interfaces	37
2.7	Evaluation metrics	39
2.8	Final considerations	40
3	WORD RECOGNITION AS SPELLING CORRECTION	43
3.1	Eye typing Edit Probabilities	43
3.2	Fast Approximate Searching Method for Eye Typing	45
3.3	Experiments	47
3.3.1	<i>Baselines</i>	47
3.3.2	<i>Dataset</i>	47
3.3.3	<i>Evaluation Methodology</i>	48
3.3.4	<i>Results</i>	49
3.4	Final considerations	53
4	FAST WORD RECOGNITION IN NOISY SCENARIOS	55
4.1	Fast Retrieval with Perfect Hashing	55

4.1.1	<i>Complexity Analysis</i>	58
4.2	<i>Linear Distance Heuristic</i>	59
4.3	<i>Experiments</i>	62
4.3.1	<i>Baselines</i>	62
4.3.2	<i>Evaluation Methodology</i>	63
4.3.3	<i>Results</i>	64
4.4	<i>Final considerations</i>	68
5	IMPROVEMENTS ON WORD RECOGNITION	69
5.1	<i>Revisiting our Word Recognition model</i>	69
5.1.1	<i>Noise model</i>	70
5.1.2	<i>Sources of error in eye-based typing</i>	71
5.1.2.1	<i>Eye-tracker measurement</i>	71
5.1.2.2	<i>Eye movements variability</i>	72
5.1.3	<i>Probabilities of Specific Edit Operations according to Noise Sources</i>	73
5.1.4	<i>Noise model Estimation</i>	75
5.1.4.1	<i>Estimation Algorithm</i>	76
5.2	<i>Experiments</i>	78
5.2.1	<i>Baselines</i>	78
5.2.2	<i>Evaluation Methodology</i>	78
5.2.3	<i>Results</i>	79
5.3	<i>Final Considerations</i>	82
6	EVALUATION ON A REAL INPUT SYSTEM	83
6.1	<i>Evaluation methodology</i>	83
6.2	<i>Interface</i>	85
6.3	<i>Collected data</i>	87
6.4	<i>Metrics</i>	88
6.5	<i>Results</i>	89
6.5.1	<i>Entry rate</i>	89
6.5.2	<i>Error rates</i>	91
6.5.3	<i>Ranking evaluation</i>	92
6.5.4	<i>NASA Task Load Assessment</i>	92
6.5.5	<i>Participant feedback</i>	92
6.5.6	<i>Interface-related actions</i>	94
6.5.6.1	<i>Deletion</i>	95
6.5.6.2	<i>Writing</i>	95
6.5.6.3	<i>Selection</i>	95
6.6	<i>Final Considerations</i>	97

7	CONCLUSIONS AND FUTURE WORK	99
7.1	Hypothesis an research questions revisited	101
7.2	Limitations of this work	103
7.3	Future work	103
7.3.1	<i>Improvements on Evaluation</i>	103
7.3.2	<i>Improvements on Interface</i>	104
7.3.3	<i>Better prediction models</i>	104
7.3.4	<i>Other issues</i>	104
	BIBLIOGRAPHY	105

INTRODUCTION

In this thesis we present a spelling correction method for recognizing words in eye-based typing systems and evaluate its impact on eye-typing sessions. In this chapter we first highlight the motivation for research and define the associated problem. Next we formulate the research hypothesis and questions. We then state the research objective followed a summary of the contributions of the work.

1.1 Context

Severely handicapped people, such as the ones with Amyotrophic Lateral Sclerosis (ALS) and Duchenne Muscular Dystrophy (DMD), face trouble expressing themselves as the disease progresses. Alternative communication approaches are necessary, such as eye gaze-based text entry (eye typing). This method consists in detecting the direction the user's eyes aim at the screen and interpreting what actions she intends to do, often using an electronic eye tracker device. The targets of selection are usually keys on virtual keyboards and screen regions that define different contextual actions. Despite the advances in eye typing, it is still possible to explore a large field of research towards the design of effective eye-based entry interfaces, including keyboard layout, key size, and which user action indicates the selection of a key.

Particularly, the selection of a key is an important issue in these systems. The first solutions to this problem consisted in using the gaze fixation for a certain predefined time as an indicative of key selection — this interval is called *dwell time* and has been used in the work by [Wobbrock *et al.* \(2008\)](#) and [Majaranta, Ahola & Špakov \(2009\)](#), among others. However, dwell-based approaches have important drawbacks. Long dwell times lead to eye fatigue and decreasing of typing speed, as observed by [Sarcar, Panwar & Chakraborty \(2013\)](#), while slow ones frustrate users due to longer response times as discussed by [Isokoski \(2000\)](#). In addition, short dwell times may result in the system misinterpreting casual gazing as key selection. To avoid these problems, many researchers have worked in dwell-free methods for eye typing, as

in the work reported by [Kurauchi *et al.* \(2016\)](#), [Pedrosa *et al.* \(2015\)](#) and [Sarcar, Panwar & Chakraborty \(2013\)](#), among others.

As observed, the reduction of dwell time leads to an increasing uncertainty about the user intents — uncertainty which is aggravated with the elimination of dwell time altogether. This problem is even worse in scenarios where the users present motor disabilities or they use affordable eye-trackers with limited hardware capabilities. An approach to deal with the recognition problem is to treat it as a spelling correction task.

An effective spelling correction approach consists in modeling the problem as the sending of a signal through a noise channel. Since the received signal s was probably distorted during transmission, the task is to fix this signal. A usual strategy is to determine which word t , among the ones in a large enough dictionary, likely corresponds to s in a feasible time and by using an acceptable space. Generally, the likelihood of t being s is estimated as inversely proportional to how many edit operations (character insertions, deletions, and substitutions as well as transposition of adjacent characters) should be applied to transform s into t . Due to time and space restrictions, the dictionary search space has to be reduced. As a result, the problem becomes to select a set of candidates t so that the search pattern s can be transformed into t using at most k edit operations. The system then ranks the candidates according to an estimate of how similar they are w.r.t. the search pattern.

The fast retrieval of candidates within a distance of up to k edit operations can be achieved by the use of indexes, such as in the Mor-Fraenkel algorithm named after the work by [Mor & Fraenkel \(1982\)](#). However, the size of these indexes grows exponentially with the dictionary size and k which, in practice, limits k to a small value. This is not a problem in the traditional scenario (based on physical keyboards) because while the occurrence of errors depends on many factors, such as the skill of the typist, in the traditional scenario a misspelling is a rare event, occurring only once every 20 to 50 words, on average as reported in the studies by [Jurafsky & Martin \(2009\)](#), [Kernighan, Church & Gale \(1990\)](#) and [Norvig \(2009\)](#). Thus, approximate search algorithms can assume a low error probability and a small number of errors *per word*. In practice, these methods are fast enough to find words in large dictionaries, even in scenarios with serious space constraints as observed by [Boytssov \(2012\)](#).

1.2 Motivation

The assumptions of low error density are not valid for very noisy input scenarios, such as eye-typing systems. In eye typing, errors are more common, with insertions more frequent than in traditional typing. This can be observed in the input stream illustrated in Table 1. This stream represents characters typed using the state-of-the-art eye-based QWERTY¹ keyboard contributed by [Pedrosa *et al.* \(2015\)](#) and detected by an inexpensive 30-Hz eye-tracker device.

¹ Named after the first six keys QWERTY on the top left letter row of a latin-script alphabet keyboard.

In the example, the user typed the word *presence* as part of a transcription experiment. Table 1 shows that as filtered string *wopezsebnde* obtained after filtering out noise by using a simple mode filter usually adopted to enhance quality as detailed by [Petrou & Petrou \(2010\)](#). This string could be transformed into *presence*, for instance, by deleting characters *w* and *o*, replacing *e* by *r* and *z* by *e*, deleting *b* and replacing *d* by *c*. This corresponds to 6 operations, a large number for current methods which have to process thousands of words in just a few milliseconds.

With high k values, approximate search methods become unfeasible and spurious matches of symbols are more likely, a scenario also observed in the task of genetic sequence alignment in the works by [Allison, Wallace & Yee \(1992\)](#) and [Boytssov \(2011\)](#). Further, no large corpus of errors is available for eye-based input systems, which makes it harder to adopt probabilistic approaches based on real use information proposed in literature, as in the contributions by [Jurafsky & Martin \(2009\)](#) and [Norvig \(2009\)](#).

Table 1 – Upon user eye typing *presence*, character stream *detected* by a low cost eye tracker and *filtered* out to remove noise – stream captured using QWERTY keyboard algorithm contributed by [Pedrosa et al. \(2015\)](#).

detected	wwwwwwwyoooooooooppppppppprrreeerereee tsszzzzzzzzssssssssssssssssseeeeeeeeeee efbbbbbbbbbbbbbbbbnnnnnnnnnnnnnnnnnnnn nnnnnvnbvcccxxdddddccccccddddd ddddweeeeeeeeeee
filtered	wopezsebnde

Source: ([HANADA et al., 2016](#))

1.3 Problem Definition

The general problem of word recognition can be stated as combined of two specific problems: (i) approximated search and (ii) candidate ranking. Below we formally define these problems.

Let a string be a finite sequence of characters over an ordered alphabet Σ , Σ^* be all strings over Σ , ϵ denote an empty string, dictionary $\mathcal{D}$ be an ordered set of strings, and $s_{[i]}$ denote the i -th character of a string. We also use this index notation to represent substrings, such that $s_{[i:j]}$ denotes a substring of s starting at the i -th character and finishing at the j -th. Concatenations are denoted by sequences of strings/characters. Vowels a and b denote characters while consonants s , t , and w , arbitrary strings.

A substring u can be transformed into substring v by means of an *edit operation* which is denoted by $u \rightarrow v$. These operations are normally restricted to character insertion ($\epsilon \rightarrow a$), deletion ($a \rightarrow \epsilon$), substitution ($a \rightarrow b$), and transposition of adjacent characters ($ab \rightarrow ba$). The sequence of edit operations which transform a string s into t is called an *edit script*. Given strings s and t , we use $\mathcal{E}_{DEL}(s, t)$ to denote the number of deletions in the optimum edit script $\mathcal{E}$

between s and t .² Similarly, we use $\mathcal{E}_{IST}(s, t)$ to denote the number of insertions, substitutions, and transpositions in $\mathcal{E}$.

Given a search pattern $s \in \Sigma^*$ and a dictionary string $t \in \mathcal{D}$:

- We define the problem of *approximated dictionary search* (selecting a set of promising candidates) as follows: Let k be the maximum number of edit operations allowed over s . The problem consists in retrieving all strings $t_i \in \mathcal{D}$, such that $\mathcal{E}_{DEL}(s, t_i) + \mathcal{E}_{IST}(s, t_i) \leq k$.
- We define the problem of *ranking the candidates* with an edit distance-based ranking function to sort the candidates so that the top-ranked candidate $\hat{w}$ is more likely to be the input word w .

It is important to observe that along, this thesis, we adopt a slightly different version of the problem of approximated dictionary search. Specifically, we define d as the maximum number of deletion operations, and k as the maximum number of insertions, substitutions, and transpositions allowed over s . We then consider approximated search as the problem of retrieving all strings $t_i \in \mathcal{D}$, such that $\mathcal{E}_{DEL}(s, t_i) \leq d$ and $\mathcal{E}_{IST}(s, t_i) \leq k$. Given these definitions, we intend to find k , d , and a ranking function which allow the design of an effective word recognizer to be used in the eye-based typing domain.

1.4 Research Hypothesis and Questions

Norvig (2009) observes that, in noisy-channel based models, effective estimates to determine how good a candidate is are based on domain knowledge about the noise sources and error distributions, including probabilities of specific edit operations. For instance, consider the misspelled word “thew” and candidate corrections “the” and “thaw”.³ Both seem to be good candidates since “thaw” would result from an accidental replacement of ‘a’ by ‘e’; and “the” is a very common English word which the typist could have misspelled by slipping its finger off the ‘e’ (since ‘w’ is an immediate neighbor of ‘e’ in a QWERTY keyboard). This knowledge on how probable are the edit operations and words can be derived from use logs and error corpora.

It is also well known that these error distributions are not the same when considering different input systems due to the influence of different error sources. To illustrate this point, Figure 1 and Table 2 show the error distribution for a transcription experiment using an eye typing system. Table 2 also presents the error distribution obtained for the list of English common errors used by Norvig (2009) and the list of errors observed on a spontaneous keystroke experiment conducted by Baba & Suzuki (2012). The keystroke experiment includes errors observed before the user self corrects her misspellings.

² Although the term *optimal edit script* has many possible meanings, we can generally interpret it as *most likely* in the sense that it corresponds to the most likely combination of edit operations in a given context.

³ Example adapted from <https://norvig.com/spell-correct.html>.

Figure 1 – Error distribution for $0 \leq k \leq 10$. As in traditional approximated search works, k corresponds to all types of errors in the string, including deletions.

Source: (HANADA *et al.*, 2016)

Table 2 – Error distribution in three different input systems. Columns labeled ‘Dels’, ‘Subs’, ‘Ins’ and ‘Trans’ stand for deletions, substitutions, insertions, and transpositions, respectively.

	Dels	Subs	Ins	Trans
Common errors	35.7	31.3	23.2	9.9
Keystroke	23.0	51.4	23.0	2.7
Eye typing	6.0	15.7	77.8	0.3

Source: (HANADA *et al.*, 2016)

As expected, while the most common errors observed with traditional input devices are deletions and substitutions (users perceive and correct substitutions more frequently, which explains the differences between the first two lines in the table), insertion errors are much more common when using a very noisy eye typing device — even when a noise filtering is carried out. Transpositions were also surprisingly uncommon. According to Figure 1, for $k = 2$, only 63% of the errors are caught in an eye-typing scenario. It is only for $k = 11$ that 99% of the errors are covered. A very different observation was reported by Norvig (2007) for traditional spelling correction: 80 to 95% of the misspellings are at edit distance 1 and about 99%, at distance 2.

These differences suggest that, to be effective, a word recognizer for eye-based typing should take into account specific error sources found in that domain. These sources are well documented in literature and include the eye tracker device, the variability of the eye movements, and particular user behaviors such as user’s predisposition to follow competitive visual stimuli, as observed by Beers (2007) and Pedrosa *et al.* (2015). In fact, an effective model should probably combine these sources along with others found in any typing system, such as those identified by Baba & Suzuki (2012) including: (i) the interface design and how familiar it is for the user; (ii) visual factors such as visual similarity of characters, position in a word, and character repetition; (iii) phonological factors such as phonological similarity of characters/words; and (iv) cognitive factors such as language domain and vocabulary familiarity.

Another interesting aspect of the prominence of insertion errors in Table 2 is that it can enable the use of very fast approaches to select candidates. The problem with fast search algorithms in the Mor-Fraenkel family is the exponential growth of their index with the dictionary size. In this family of methods, all edit operations are represented just by deletions in the search pattern s and in the dictionary words t . For instance, $s = \text{kind}$ can be matched with $t = \text{bind}$ by substituting k by b . This substitution is equivalent to delete k in s and b in t , which results in the match of the residual strings ind .

These residual strings and corresponding deletion lists are stored in deletion indexes (Mor-Fraenkel indexes) which allow fast retrieval of candidates. Due to the Mor-Fraenkel index design, the growth of dictionary deletion lists is caused by deletion, substitutions, and transposition errors in the search pattern. Since insertion errors are the commonest in eye typing, it is possible to handle more insertion errors than other errors without increasing the index size. To accomplish this, the index should be asymmetrical, allowing a larger number of deletions in the search pattern than in the dictionary word.

In sum, we can probably obtain an effective word recognizer for eye-based typing by using domain-specific error information. We can also take advantage of the particular error distribution in this domain to design an asymmetrical Mor-Fraenkel method which can enable fast selection of candidates under reasonable space restrictions. Such a method can improve user performance since it will enable the recognition of more words and will diminish the time the user spends looking for the correct suggestion. Based on such motivation, we can formulate our main hypothesis in this thesis:

Hypothesis: *A ranking based on a noise model that takes advantage of the typical noise in eye typing can improve user performance and provide a better user experience. Although this ranking is more complex and therefore has a greater computational cost, it is possible to implement it in a viable way by taking advantage of the distribution of errors observed in eye typing.*

Given the presented hypothesis, the following questions arise:

- How could the noises specific to the eye-based typing domain, in particular when considering affordable eye trackers, be modeled to improve word recognition?
- How to combine eye-based typing specific noises with general noise observed in any typing system to design an effective word recognition model?
- How could these models be efficiently estimated, in real time, so that they are feasible in a real interface?
- What is the practical impact of these models on the user experience when evaluated in a real interface? Do they help the user to achieve faster typing speeds without losses in accuracy? Do they provide a more enjoyable use experience?

In this work, we intend to provide answers to such questions. The pursuit for these answers defined our objectives, described in next section.

1.5 Objectives

Our aim is to define a method to improve the capability of eye-based typing systems to recognize typed words. This method should be computationally efficient and effective to be viable in a real input system. This main objective translates into the following specific objectives:

- Determine how specific noises, such as those derived from eye tracker devices and the natural eye movement variability, affect eye-based typing and how such noises could be modeled to improve recognition when compared with current state-of-the-art methods.
- Propose models to combine noise derived from eye specific error sources with those found in any typing domain to create a better word recognizer.
- Propose algorithms and processes able to effectively estimate these models, so that they can be used in a real interface. More specifically, these models have to provide high quality recognition with acceptable time and space restrictions.
- Evaluate the practical impact of the proposed models on the user's typing experience, verifying if they lead to faster and accurate typing, contributing for a pleasant use experience when compared with a state-of-the-art approach.

1.6 Contributions

In this thesis we present a spelling correction method for recognizing words in eye-based typing systems and evaluate its impact on eye-typing sessions. Our method includes an approximate search component which selects candidates and a ranking component which orders them to suggest the most likely typed word. The search component is based on an asymmetric Mor-Fraenkel index. The ranking component adopts a noisy-channel based model to find the most probable words.

In summary, the contributions made during this research are:

- A variant of a Mor-Fraenkel approach able to deal with very noisy inputs without an unfeasible growth of the index size. Although, in this thesis, this method was applied to eye typing, we argue that it can be used in other noisy input scenarios.
- Very accurate likelihood estimators which combine general and specific domain knowledge about typing errors.
- Very fast search mechanisms able to provide suggestions within a predefined time limit.
- A study on how to balance the influence of the language and noise components of the noisy-channel based model.

- A study on how to include information associated with the eye movements, to better capture errors related to movement variability.
- A study on the input stream pre-processing techniques to filter unintended selections.
- A study on the impact of the proposed ranking on the users' behavior while they are typing with eyes.
- Comprehensive evaluations of the model and their constituent components using real-use log data and real eye-typing sessions.
- Detailed data logs of the experiments performed with groups of users from the University of São Paulo and from the University of Toronto.⁴ These groups correspond to non-native and native English speakers, respectively.

Part of these contributions were reported in two manuscripts published in the Proceedings of the ACM Conferences on Information and Knowledge Management (CIKM):

- “Effective Spelling Correction for Eye-based Typing using domain-specific Information about Error Distribution” (HANADA *et al.*, 2016)
- “Fast Word Recognition for Noise channel-based Models in Scenarios with Noise Specific Domain Knowledge” (CRISTO *et al.*, 2017)

The complete study presented in this thesis is reported in the manuscript “A Noisy-channel based Model to Recognize Words in Dwell-free Eye Typing Systems” submitted to the ACM Transactions on Computer-Human Interfaces (TOCHI), and is currently under review.

Also related to this thesis, we have collaborated in the study of an alternative interface for eye-typing. Unlike the interface presented here, that one was not based on a QWERTY layout and focused on the Brazilian Portuguese language. The work was presented at the 35th ACM International Conference on the Design of Communication (SIGDOC):

- “A word-prediction eye-typing approach for Brazilian Portuguese entries using geometric movements” (RAMOS *et al.*, 2017).

During the doctorate, we also collaborated with other researchers in topics associated with modeling and interfaces, but without direct relation to this thesis. All these works were published in the Webmedia conference:

- “A Platform for the Recommendation of Points of Interest in Brazilian Cities: Architecture and Case Study” (DOMINGUES *et al.*, 2015)
- “How do metrics of link analysis correlate to quality, relevance and popularity in wikipedia?” (HANADA; CRISTO; PIMENTEL, 2013)
- “An investigation of the relationship between the amount of extra-textual data and the quality of Wikipedia articles” (HIMORO *et al.*, 2013).

⁴ Datasets available in <<https://github.com/raizahanada/eye-typing-data>>

All these works were partially funded by the Google Research Awards in Latin America, 2016 and 2017, and during four years by Amazon Research Foundation (FAPEAM).

1.7 Thesis Organization

This thesis is organized as follows. Chapter 2 provides the background required for understanding the proposal, the notation used along the text, and related work. Chapter 3 describes and evaluates our first noisy model, AMMLSEP, which combines noise derived from general and specific error sources. It also introduces an asymmetrical Mor-Fraenkel approach implemented using a Trie. Chapter 4 describes and evaluates our second and third models, AMPH-MML and AMPH-LHD. They extend AMMLSEP by replacing the Trie with a Minimal Perfect Hashing, introducing a linear heuristic for estimating the noise model and adopting pruning approaches which ensure that response time will not be longer than a certain limit. Chapter 5 describes and evaluates C-AMPH-LHD, an improved version of AMPH-LHD. Its improvements are derived from studies on pre-processing techniques, how to balance the noise and language components of the proposed recognition model, and how to extend the noise model to include errors derived from the eye movements variability. Chapter 6 evaluates C-AMPH-LHD in a real eye-typing task. Finally, Chapter 7 concludes with perspectives for future work.

BACKGROUND AND RELATED WORK

The detection of a message from a corrupted source and related problems have been part of a diverse set of applications such as spelling correction as reported by [Mays, Damerau & Mercer \(1991\)](#), query prediction as in the work by [Baytin *et al.* \(2013\)](#), optical character recognition, spoken recognition, image alignment, database search and in the research by [Gollapudi & Panigrahy \(2006\)](#), and alignment of biological sequences as reported by [Allison & Yee \(1990\)](#). As a result, a large body of literature addresses such subjects.

Focusing on topics directly related to the problem tackled in this thesis, in this chapter we summarize the main concepts regarding edit-based distance metrics for candidate ranking and approximate search, errors associated with eye typing, and review related work. We also introduce the notation to be used along this text.

2.1 Errors associated with eye-typing systems

Spelling correction for very noisy input systems share most of the same causes of errors with other input systems. [Baba & Suzuki \(2012\)](#) identified some of these causes: (i) *physical factors* such as motor control of hands and fingers and distance between the keys; (ii) *visual factors* such as visual similarity of characters, position in a word, and same character repetition; and (iii) *phonological factors* such as phonological similarity of characters/words. Based on the observations by [Jurafsky & Martin \(2009\)](#), we also include (iv) *cognitive factors* such as language domain and vocabulary familiarity.

All factors besides motor control of hands and fingers also affect eye-typing systems. However, while motor control of hands and fingers are specific for hand-based input, a range of error factors are specific for eye-based typing. These errors are associated with aspects that include user behaviour, eye movements, eye tracker device and software interface. Thus, in addition to the visual, phonological, and cognitive factors, we include the variability eye

movements, measurements by the eye tracker device, and interface characteristics and user behaviour, as detailed next.

- *Eye movements variability*: The eye movements most relevant for typing are saccades and fixations. Saccades are fast ballistic eye movements which allow the eye to focus on its intended target, for instance, the next key to be typed. During such movements, no visual processing takes place. The target is only properly captured by the visual system during the fixation, when the eye stays almost inert. While eye fixation is normally modelled using a simple Gaussian component, variability of eye saccades are more complex to model.

According to [Beers \(2007\)](#), the main causes of saccade variability are (i) sensory noise and (ii) noise associated with movement planning and execution (*motor noise*). Sensory noise results from uncertainty in the localization of a target out of fovea region, due to low density of cells in retina: it can be estimated using models for vision acuity, with larger errors for larger distances and oblique movements. [Beers \(2007\)](#) observed that sensory noise explains 57% of the variance in saccade endpoints, whereas motor noise explains 42%. It is important to observe that such noise increases the possibility that the eye misses the visual target and points out to a position in its neighborhood.

- *Eye tracker measurement*: As observed by [Beers \(2007\)](#), for accurate trackers and in well controlled conditions, eye tracker measurement noise is negligible. However, this is not the case for inexpensive eye trackers. This noise varies according to the accuracy and precision of the device. The accuracy is the average difference between the real stimuli position and the measured gaze position. Precision is the ability of the eye tracker to reliably reproduce the same gaze point measurement. Accuracy, in particular, depends on the user, environment illumination, stimuli, calibration, and the position of the eyes and head.
- *Interface characteristics and user behaviour*: The interface design and the familiarity of the user with the interface determines the eye movements required by users to achieve their goals. In particular, keyboard layout and space occupied by the interface defines how close are the keys to each other.

The eyes are also intrinsically associated with cognitive processes such that any competitive stimulus can grab user attention and produce undesirable eye movements. As observed by [Pedrosa et al. \(2015\)](#), when a user does not remember the exact position of the next key, she engages into a search behavior which leads to a sequence of saccades and fixations hardly distinguishable from regular typing movements. As a result, a sequence of errors is captured by the tracker. A distinct characteristic of such burst of errors is the importance of the context, since the next wrong key has a high probability of belonging to the neighborhood of the current one.

2.2 Preliminary Notation

In this thesis, we follow the notation used by Boytsov (2011) briefly presented in Chapter 1. Let Σ be a finite ordered alphabet of size $|\Sigma|$. A string is a finite sequence of characters over Σ . The set of all strings over Σ is represented by Σ^* . The set of all strings in the vocabulary of a dictionary is denoted by $\mathcal{D}$. An empty string is denoted by ε . We use letters a and b to represent characters and u , v , and w to represent arbitrary strings. The letters s and t represent the search pattern and the candidate dictionary word, respectively. The length of a string $s \in \Sigma^*$ is denoted by $|s|$. A series of strings/characters represents its concatenation. We use the typewriter font to indicate literal characters such as `example`. The i -th character of a string is represented by $s[i]$. A substring starting at the i -th character and finishing at the j -th character is denoted by $s[i:j]$. If the first index is greater than the second, the substring is oriented backwards. Thus, the reverse of $s[i-1:i]$ is $s[i:i-1]$. Additional notation, presented in this chapter, is summarized in Table 3.

Table 3 – Notation and symbols used in this thesis

Notation	Meaning
Σ	Finite ordered alphabet of size $ \Sigma $
Σ^*	Set of all strings over Σ
$\mathcal{D}$	All strings in a dictionary vocabulary
ε	Empty string
a, b	Any characters
s, t	Search pattern and candidate dictionary word
u, v, w	Arbitrary strings
sw	Concatenation of strings s and w
swa	Concatenation of string sw with character a
$ s $	Length of a string $s \in \Sigma^*$
$s[i]$	i -th character of s
$s[i:j]$	s substring from $s[i]$ to $s[j]$
$s[j:i]$	Reverse of $s[i:j]$
<code>example</code>	Literal characters
$u \rightarrow v$	Edit operation which transforms u into v
$\prec$	Start of the word
$\varepsilon \rightarrow a$	Character insertion
$a \rightarrow \varepsilon$	Character deletion
$a \rightarrow b$	Character substitution
$ab \rightarrow ba$	Transposition of adjacent characters
$\delta_{u \rightarrow v}$	Cost of edit operation $u \rightarrow v$
$\mathcal{E}(s, t)$	<i>Edit script</i> : sequence of edit operations which transform s into t
$\mathcal{E}_{DEL}(s, t)$	Number of deletions in $\mathcal{E}(s, t)$
$\mathcal{E}_{IST}(s, t)$	Number of insertions, substitutions, and transpositions in $\mathcal{E}(s, t)$

2.3 Noisy channel theory

Noisy channel models have been widely studied since they are used in many applications as spelling correction and genetic sequences alignment. For instance, in spelling correction, mistyping a string w producing s can be seen as the transmission of w through a noisy channel as in the work by [Kernighan, Church & Gale \(1990\)](#) and [Mays, Damerau & Mercer \(1991\)](#). This model assumes that the transmission of a signal w , originated by the sender, can be modified during transmission in a noisy channel resulting in signal s . To recover the original signal the receiver has to determine, among possible candidates t in a vocabulary $\mathcal{D}$, which one is more likely to be the distorted w . Using Bayesian inference the best hypothesis $\hat{w} \in \mathcal{D}$ is given by Eq. 2.1:

$$\hat{w} = \arg \max_{t \in \mathcal{D}} P(t|s) = \arg \max_{t \in \mathcal{D}} \eta P(s|t)P(t) \quad (2.1)$$

where s is the input pattern, $t \in \mathcal{D}$ is a dictionary word, η is a normalization constant, $P(s|t)$ indicates how likely s could be a distorted version of t and $P(t)$ is the prior probability of t . The application of this model for spelling correction is straightforward since the likelihood $P(s|t)$ can be estimated by a similarity metric of the observed input stream s and the dictionary word t . In that case, $P(t)$ can be viewed as a language model for t which indicates its prior probability considering a certain language and word context.

2.4 Candidate ranking

Considering that we have a set of all the candidate words $t \in \mathcal{D}$ given an entry input s , our goal is to find out how the candidate words should be ranked in order to find which one is most likely to be the intended word w . The most common metrics used to this end are based on the edit operations necessary to transform s into t . Previous methods have focused mostly on the Levenshtein distance and on the Damerau-Levenshtein distance – named after the work by [Levenshtein \(1965\)](#)¹ and [Damerau \(1964\)](#). However, other distances have also attracted interest as attempts to properly define the best alignment based on information extracted from data corpora. We present below several metrics that can be applied in this task.

2.4.1 Classic edit distances

When solving Eq. 2.1, the distance metric used to estimate $P(s|t)$ is normally based on the fact that a string can be transformed into another by a set $\mathcal{E}$ of atomic substring transformations e . $\mathcal{E}$ is called an *edit script* while e is a *basic edit operation*. A basic edit operation which maps u in v is represented by $u \rightarrow v$. Basic edit operations are generally restricted to character

¹ As translated from the original Russian manuscript by [Levenshtein \(1966\)](#)

insertion ($\epsilon \rightarrow a$), character deletion ($a \rightarrow \epsilon$), character substitution ($a \rightarrow b$), and transposition of adjacent characters ($ab \rightarrow ba$). Note that edit operations can overlap so that a single substring can be modified twice. Usually such overlapping is not allowed so as to diminish computational complexity, which results in *restricted edit scripts*.

We can now define *edit distance* $ED(s, t)$ as the length of the shortest (optimal) edit script that transforms s into t . The edit distance which includes character insertion, deletion, and substitution is referred to as Levenshtein distance. If transpositions are also included, the distance is referred to as Damerau-Levenshtein. These are named after the work by [Damerau \(1964\)](#) and [Levenshtein \(1966\)](#). Thus, given that the cost of an edit operation is given by $\delta_{s \rightarrow t}$, the cost of the edit script $\mathcal{E} = s_1 \rightarrow t_1, s_2 \rightarrow t_2, \dots, s_{|\mathcal{E}|} \rightarrow t_{|\mathcal{E}|}$ is given by $\delta(\mathcal{E}) = \sum_{i=1}^{|\mathcal{E}|} \delta_{s_i \rightarrow t_i}$.

The efficient computation of the Damerau-Levenshtein distance corresponds to finding an *optimal alignment*. An alignment is defined as follows. Let strings s and t be partitioned in the same number of possible empty substrings $s = s_1 s_2 \dots s_\ell$ and $t = t_1 t_2 \dots t_\ell$ such that s_i and t_i cannot be empty at same time. We say that partition $A = (s_1 s_2 \dots s_\ell, t_1 t_2 \dots t_\ell)$ is an alignment between s and t where substring s_i is aligned with substring t_i . This alignment represents the edit script $\mathcal{E} = s_1 \rightarrow t_1, s_2 \rightarrow t_2, \dots, s_\ell \rightarrow t_\ell$. Thus, the cost of alignment A is given by $\delta(\mathcal{E})$ and the optimum alignment is the alignment with the minimum cost.

Assume that the cost is only a type of selection score. The score of the optimum alignment between s and t can be obtained recursively from the score of their prefixes, using a dynamic programming approach. Let $s_{[1:i]}$ and $t_{[1:j]}$ be the prefixes of i -th and j -th characters of s and t respectively. Let $A = (s_1 s_2 \dots s_\ell, t_1 t_2 \dots t_\ell)$ be the optimum alignment between $s_{[1:i]}$ and $t_{[1:j]}$ with score $C_{i,j}$. Let $\chi: \mathbb{R}^m \rightarrow \mathbb{R}$ be a function that combines m input values and $\psi: \mathbb{R}^m \rightarrow \mathbb{R}$ be a function that selects one among m input values. Considering the edit operations of Damerau-Levenshtein edit distance, $C_{i,j}$ can be computed by Eq. 2.2:

$$C_{0,0} = \zeta \tag{2.2}$$

$$C_{i,j} = \psi \begin{cases} \chi(C_{i-1,j}, \delta_{s_{[i]} \rightarrow \epsilon}) & \text{if } i > 0 \\ \chi(C_{i,j-1}, \delta_{\epsilon \rightarrow t_{[j]}}) & \text{if } j > 0 \\ \chi(C_{i-1,j-1}, \delta_{s_{[i]} \rightarrow s_{[i]}}) & \text{if } i, j > 0, s_{[i]} = t_{[j]} \\ \chi(C_{i-1,j-1}, \delta_{s_{[i]} \rightarrow t_{[j]}}) & \text{if } i, j > 0, s_{[i]} \neq t_{[j]} \\ \chi(C_{i-2,j-2}, \delta_{s_{[i-1:i]} \rightarrow s_{[i-1:i]}}) & \text{if } i, j > 1, s_{[i-1:i]} = t_{[j-1:j]} \end{cases}$$

where ζ is a constant. While this approach was proposed several times in different domains, the general formulation shown in Eq. 2.2 is due to [Powell et al. \(1998\)](#). In particular, the restricted Damerau-Levenshtein edit distance, as proposed by [Wagner & Lowrance \(1975\)](#), can be obtained by setting parameters $\zeta = 0$, $\psi = \min$, and $\chi = +$. As this distance measures dissimilarities, $\delta_{s_{[i]} \rightarrow s_{[i]}} = 0$, while $\delta_{s_{[i]} \rightarrow \epsilon} = \delta_{\epsilon \rightarrow t_{[j]}} = \delta_{s_{[i]} \rightarrow t_{[j]}} = \delta_{s_{[i-1:i]} \rightarrow s_{[i-1:i]}} = 1$.

By carefully studying the parameters of Eq. 2.2, one can obtain an estimate for $P(t|s)$ in Eq. 2.1. Thus, to completely solve Eq. 2.1, it is only necessary to retrieve all candidates $t \in \mathcal{D}$.

2.4.2 Related distance metrics

We find in later work other metrics to calculate the edit distance that can vary according to the purpose in which they are applied. An important probabilistic interpretation of the distance was given by Allison & Yee (1990) when addressing the problem of finding the most probable alignment of two genetic sequences. Following an interpretation from the Information Theory area, they model the alignment of two sequences as the hypothesis that the better its mutual compression, the better its alignment. To calculate how well the string is compressed, they provide a probabilistic interpretation for Eq. 2.2 by setting $\zeta = 1$, $\psi = \max$, and $\chi = \times$. Costs are reinterpreted as edit operation probabilities. Thus, given the probabilities for matching (P_m), substitution (P_s), insertion (P_i) and deletion (P_d), the following estimates are used: $\delta_{s[i] \rightarrow \epsilon} = P_d P(s[i])$; $\delta_{\epsilon \rightarrow t[j]} = P_i P(t[j])$; $\delta_{s[i] \rightarrow t[j]} = P_s P(s[i], t[j] | s[i] \neq t[j])$. Transpositions are not taken into account as they are of less importance in biological sequences, and matches are proportional to how likely the matched symbols are, i.e., $\delta_{s[i] \rightarrow s[i]} = P_m P(s[i])$. However, given that in these applications errors are common and strings are large, estimates quickly become very small, leading to underflow. To solve this problem, probabilities are replaced by their negative logarithms. The adoption of logarithms allows one to interpret Eq. 2.2 according to Information Theory. This way, the alignment of two sequences, s and t , can be viewed as the hypothesis that s and t are related. We can verify how good the alignment is (and if it is a valid hypothesis) by how much it compresses the pair of sequences. To accomplish this, Eq. 2.2 is now modified by setting $\zeta = 0$, $\psi = \min$, and $\chi = +$. Given probabilities P_m , P_s , P_i and P_d , as before, costs are estimated as: $\delta_{s[i] \rightarrow \epsilon} = -\log P_d - \log P(s[i])$; $\delta_{\epsilon \rightarrow t[j]} = -\log P_i - \log P(t[j])$; $\delta_{s[i] \rightarrow t[j]} = -\log P_s - \log P(s[i], t[j] | s[i] \neq t[j])$, and $\delta_{s[i] \rightarrow s[i]} = -\log P_m - \log P(s[i])$. This approach is called Minimum Message Length (MML) Optimal Alignment and has been presented by Allison, Wallace & Yee (1992) and Allison & Yee (1990).

Other probabilistic data-oriented approaches have been proposed in the context of spelling correction. Because these works, in general, estimate the most probable alignment based on its corresponding edit script, matches are ignored and only estimates for deletions, insertions, substitutions, and transpositions are provided. Church & Gale (1991) extended the noisy-channel model, using Damerau-Levenshtein distance, by allowing each *specific* edit (e.g. the substitution of a by b) to have a probability associated with it. Further, insertion and deletion probabilities are conditioned on context. Later, Brill & Moore (2000) extended the noisy-channel model to support weighted substitutions of arbitrary substrings, achieving higher correction accuracy. Norvig (2009) advocates that the probability of each specific edit operation should be derived from large corpora, such as the list of the most common errors. By using such information, the model would capture most of the cognitive and mechanical errors resulting from inconsistencies

Figure 2 – Trie for keys I, in, inn, tea, ten, and to.

Source: (HANADA *et al.*, 2016)

in orthography, phonetic similarities, lack of familiarity with vocabularies and physical proximity among the keys due to keyboard layouts, for instance. Norvig’s proposal is equivalent to the following parameter settings in Eq. 2.2: $\zeta = 1$; $\psi = \max$; $\chi = \times$; $\delta_{s[i] \rightarrow \varepsilon} = P(s[i] \rightarrow \varepsilon)$; $\delta_{\varepsilon \rightarrow t[j]} = P(\varepsilon \rightarrow t[j])$; $\delta_{s[i] \rightarrow t[j]} = P(s[i] \rightarrow t[j])$, and $\delta_{s[i-1:i] \rightarrow s[i:i-1]} = P(s[i-1:i] \rightarrow s[i:i-1])$. Regarding its implementation, the spell checker proposed by Norvig uses a neighborhood generation method based on hash tables. For a complete review on spelling correction based on noisy-channel models, we refer to the reader Jurafsky & Martin (2009).

2.5 Approximated dictionary search

Besides understanding how to rank candidate words, we need to resolve how to find them. The process of selecting a set of promising candidates $t \in \mathcal{D}$, given input string s , is called *approximated dictionary search*. Given a search pattern $s \in \Sigma^*$ and a dictionary string $t \in \mathcal{D}$, the problem consists in retrieving all strings $t_i \in \mathcal{D}$, such that $\mathcal{E}_{DIST}(s, t_i) \leq k$ and k is the maximum number operations (deletions, insertions, substitutions, and transpositions) allowed over s .

Assuming a reasonable number of candidates (for instance, thousand words of a dictionary), a fast retrieval process has to be devised. Many of the possible approaches look for the candidates using data structures such as hash tables and tries. A trie (or a prefix tree) is an ordered tree in which each edge is associated with a label. It is called a complete trie if each label represents a character and all characters in key s are represented in the trie. Each node θ is associated with a string formed by the concatenation of all characters on the way from the root to θ . Thus, all the descendants of a node have a common prefix of the string associated with that node and the root is associated with an empty string. Because of that, tries can answer prefix queries in a straightforward way. Figure 2 illustrates a trie. We call lookup the action of looking for a possible candidate in its storage structure. Thus, the lookup cost of a trie is proportional to $O(|s|)$ in the worst case.

A hash table maps keys to values by using a hash function to compute an index for the key s into an array of slots. An ideal hash function should map each key to a unique slot. However, this is not always possible since there may be collisions. In average, a well-designed hash table used for dictionary search has lookup cost proportional to $O(|s|)$. When the hash table

has no collisions it is called Perfect Hashing and its lookup cost is $O(1)$.

2.5.1 Related approximate searching methods

Many index-based methods have been proposed to approximate searching. [Boytsov \(2011\)](#) contributes a thorough review with an extensive comparison among the most important methods. Following his taxonomy, indexing methods for approximate searching can be classified according to how they deal with the input pattern. *Direct methods* look for the entire pattern; (ii) *sequence-based filtering methods* process string representations derived from fragments of the strings; and (iii) *hybrid methods* can combine different approaches.

The most important direct methods are prefix trees, pattern-partitioning methods, space metric methods, vector-space methods, neighborhood generation methods, and combinations of these methods. Prefix trees (Tries) and space metric methods reduce the sets of words to be searched by partitioning them to create a hierarchy of space partitions. The hierarchy is recursively traversed according to a proximity metric, such that the search focuses on increasingly smaller sets of words. Pattern-partitioning methods filter out strings, looking for string fragments, to get a small number of candidates who are then checked to avoid false positives. Vector-space methods are similar to pattern-partitioning methods, except they look for string vector representations instead of fragments. Neighborhood generation methods process the input string by applying all possible combinations of up to k edit operations such that all the dictionary words within a Damerau-Levenshtein distance of k are generated. Since the size of a k -neighborhood is $O(|s|^k|\Sigma|^k)$, mainly due to insertion operations, this approach is only practical for small values of k , $|s|$ and $|\Sigma|$, as observed by [Boytsov \(2012\)](#). Sequence-based filtering methods first filter out words $t \in D$ such that $ED(s, t) > k$. To this end, they look directly for string fragments (*pattern-partitioning methods*) or vector representations of the string (*vector-space methods*). After filtering out, they check the remaining candidates to avoid false positives.

As observed by [Boytsov \(2012\)](#), the retrieval time of all methods grows exponentially as k increases, resulting in performance equivalent to sequential searching. Sequence-based filtering methods are not very fast and so are space metric methods. For natural language data, methods based on string tries and neighborhood generation (reduced alphabets) are the best. In particular, the hybrid method FB-trie proposed by [Mihov & Schulz \(2004\)](#), which combines prefix tress (Tries) with pattern partitioning, achieved the best trade-off between time and space efficiency. This method uses two tries to satisfy a query. The first trie is a regular trie while the second is built over reversed strings. The search algorithm divides the pattern into two parts and searches for the first part in the regular trie (with $t \leq k/2$ errors) and for the reverse of the second part in the reversed trie (with $k - t$ errors).

The only method to consistently outperform FB-Tries in time efficiency was a neighborhood generation approach, the Mor-Fraenkel method named after the work by [Mor & Fraenkel \(1982\)](#). However, given its large space requirements, it is limited to small dictionaries and small

k. It is based on the idea that all the edit operations over a search pattern can be represented only by deletions. This is due to the fact that substitutions and transpositions can be represented by means of insertions and deletions while an insertion in the search pattern is equivalent to a deletion in a dictionary string. For instance, $s = \text{hous}$ can be matched with $t = \text{house}$ ($t \in \mathcal{D}$) by either inserting an *e* at the end of s or deleting *e* from the end of t . Considering that deletion-only neighborhoods can provide all necessary information, at index time, the method obtains the deletion-only k -neighborhoods² of every dictionary string t . The resulting k -residuals³ of every t are stored in the index, along with their deleted characters and respective positions. At search time, instead of generating the full neighborhood of s , the method obtains the much smaller deletion-only k -neighborhood. The s residuals are then exactly searched in the index.

Extensions to this basic idea avoid storing additional information, deal with false positives directly by checking ambiguous matches, and use more efficient ways to store the set of residual dictionary strings. This is the case of FastSS, by [Bocek, Hunt & Stiller \(2007\)](#), and Symspell, by [Garbe \(2012\)](#). In particular, [Bocek, Hunt & Stiller \(2007\)](#) and [Garbe \(2012\)](#) proposed clever ways to use Mor-Fraenkel indices without storing residual strings. In a following work, [Boytssov \(2012\)](#) proposed another Mor-Fraenkel variant where deleted characters and positions were indexed using a perfect hash function computed for a set of residual dictionary strings. The proposed method used 4-8 times less space without time performance degradation when compared to the traditional implementation. [Chegrane & Belazzougui \(2013\)](#) presented an algorithm as efficient in search time and index space as the proposal by [Boytssov \(2012\)](#), and with a more time and space-efficient building algorithm. Their algorithm, however, does not support transpositions.

2.6 Eye-typing interfaces

With the intention of improving the speed and comfort of eye-based typing, alternative keyboards have been studied in literature, as it is the case with gaze-based input. [Ward & MacKay \(2002\)](#) defined a new paradigm of eye-based typing when proposed a gaze-based approach as an extension to their language model-based entry input method. Moving away from QWERTY, their proposal is based on the dynamic presentation of the most probable letters depending on the ones previously entered.

Some methods, such as the one by [Isokoski \(2000\)](#), are designed to minimize the selection of unwanted keys and conserve the original display area by tracking the user's eye movements between off-screen targets and translating them to the intended input. In a similar approach, [Tula, Campos & Morimoto \(2012\)](#) proposed the Dynamic Context Switching which changes the information presented on the screen depending on the previous dwell-based selected input.

² The set of strings, derived from s , by applying up to k deletions. For instance, the 1-neighborhood of bind is $\{\text{bind}, \text{ind}, \text{bnd}, \text{bid}, \text{bin}\}$.

³ Strings obtained by the deletion of up to k characters. For instance, the 1-residuals of bind are bind , ind , bnd , bid , and bin .

Other studies adopt different keyboard layouts to enable multiple selection, avoid collisions and provide less exhaustive interfaces. [Bee & Andre \(2008\)](#) present Quikwriting, an approach that include a circular keyboard. The evaluation of this method resulted in a typing rate of 5 Words per Minute (WPM)⁴ including only non-experienced users. EyeWrite, contributed by [Wobbrock et al. \(2008\)](#), is an interface based on predefined gestures to generate characters: the authors report that, after 14 sessions, novice users could achieve up to 5 WPM. Gesture-based systems have usually a long learning curve, so experienced users would have a much higher typing rate. Other keyboard layouts exploit geometric movements as in the work by [Ramos et al. \(2017\)](#), and in key arrangements based on the frequency of letters as in the proposal by [Antonio et al. \(2014\)](#).

The recent work by [Topal, Benligiray & Akinlar \(2017\)](#) has combined much of these previous ideas creating a gaze-based circular layout where the size of a zoomable area depends on the probability of the next letter. In a similar approach, [Huckauf & Urbina \(2008\)](#) combine gaze-based input and dwell time to support eye-based typing using pie-based keyboard. Despite the variety of studies on alternate layouts, many works and commercial products still focus on the familiar QWERTY layout.

The biggest problem when using QWERTY keyboards is that the proximity and arrangement of the letters can lead the programs to misinterpret the keys that users gaze at. The most famous problem is called Midas' effect in which the methods consider as part of the entry all regions gazed by users. To avoid that effect researchers use approaches that include to define a gaze interval called *dwell time* — an approach that has been widely used in literature as in the research by [Diaz-Tula & Morimoto \(2016\)](#), [Tula, Kurauchi & Morimoto \(2013\)](#), [Majaranta, Ahola & Špakov \(2009\)](#) and [Wobbrock et al. \(2008\)](#).

One challenge of dwell-based approaches is the definition the interval since the larger the interval, the larger the eye fatigue and, conversely, the smaller the typing efficiency. To provide fast eye typing, recent approaches usually attempt to decrease or eliminate the dwell time which results in increasing uncertainty about the user actions and more recognition errors.

A remarkable tool is EyeK proposed by [Sarcar, Panwar & Chakraborty \(2013\)](#). To predict the next character according to a previous one, EyeK highlights the most likely next characters as the user types one. This approach could achieve up to 8.8 WPM. Using prediction as well, alongside with a gaze-specialist user interface not based on a QWERTY keyboard, [Ward & MacKay \(2002\)](#) proposed Dasher, a solution that only shows the possible new sequence of symbols according to the previous inserted ones. If the predictions are accurate, the most likely next symbols are highlighted correctly, and the typing rate can be noticeably increased. The typing rates of this technique, within an hour of practice, were up to 14 WPM, while experienced users could perform up to 23 WPM.

⁴ From now on, we use the acronym WPM.

To improve speed without sacrificing accuracy, some methods adopt prediction models to reduce uncertainty. For instance, the method proposed by [Pi & Shi \(2017\)](#) adjusts the dwell time according to the probability of the next letter such that more likely sequences will require less time to be recognized. [Mott et al. \(2017\)](#) improved on this idea by slowly decreasing the dwell time to maintain a smooth typing rhythm and cause little disruption to users' typing cadence.

In order to try to eliminate the dwell time and still overcome the uncertainty about user actions, [Pedrosa et al. \(2015\)](#) proposed a method, named *Filtaryedping*, where a fixed, but very small dwell time is adopted. However, the user is not required to wait for a selection feedback before moving her eyes to the next letter. In this way, she can swipe through the letters, enabling an almost dwell-free typing. To recognize the typed word, *Filtaryedping* first smooths the sequence of letters seen by the user (scanpath) using the method proposed by [Kumar, Winograd & Paepcke \(2007\)](#). It then deletes letters from the scanpath by applying some selection heuristics yielding a search pattern s . All dictionary words t whose letters are present in s , in the same order and possibly repeated, are sorted according to the ranking function given by Eq. 2.3:

$$\text{score}(t) = \log_{10}(f_t) + 1.08 |t| \quad (2.3)$$

where f_t is the word frequency in a general corpus. In their experiments, participants without motor disabilities achieved a performance of about 15.95 WPM, while participants affected by ALS or DMD achieved an average rate of 7.60WPM.

[Kurauchi et al. \(2016\)](#) proposed a completely dwell-free technique, *EyeSwipe*. In their approach, after selecting the first character of a word in a particular way (called “reverse crossing”), the user can quickly swipe through the middle words, finishing the typing with a reverse crossing at the last character. With first and last characters reliably recognized, *EyeSwipe* is able to infer the remaining ones even if the scanpath is obtained from a quicker, and much noisier, typing. In the experiments conducted to evaluate *EyeSwipe*, the users achieved 11.7 WPM. Both approaches outperformed dwell-based approaches.

2.7 Evaluation metrics

In this thesis we use two main metrics to evaluate the developed methods. The first one is the Mean Reciprocal Rank (MRR) as defined by [Voorhees \(1999\)](#), which we use to estimate the ranking quality of the models we will present throughout the Chapters 3, 4 and 5. The second one is the WPM, which we use to compare typing speeds using our best model and a state-of-the-art method, *Filtaryedping* (cf. Chapter 6). These metrics are defined as follows:

- *MRR*: It is the mean of the inverse of the ranks at which the correct word was retrieved for a set of search patterns. Thus, given an input s_i , if correct word w was ranked at position r by method M , M is scored $RR_i = 1/r$. If the correct word was not part of the list of

suggestions given by M , M is scored 0. If w was tied with other words $w_j \in \mathcal{W}$, its score is averaged over the reciprocal ranks for all $w_j \in \mathcal{W}$. Given n search patterns, the MRR is given by $1/n \sum_i RR_i$. This metric awards words ranked near the top of the suggestion list which is appropriate for problems where there is no much space to show suggestions.

- **WPM:** It is often used to measure typing speed. WPM estimates how many words a person can type in a minute through the division of the amount of words one has typed by the amount of minutes she has spent in the task.

2.8 Final considerations

In this chapter we presented the main problems addressed by our input recognition model and the main approached used researchers to solve them. In addition, we presented a brief review of the literature on eye typing tools and the main problems faced in developing them.

Table 4 summarizes the information about the methods we tested in this thesis as word recognizers. It shows the methods designed for spelling correction proposed by [Bocek, Hunt & Stiller \(2007\)](#) (FastSS), [Norvig \(2009\)](#), [Boytsov \(2012\)](#), and [Garbe \(2012\)](#); and the methods designed for eye-typing word recognition proposed by [Pedrosa et al. \(2015\)](#) and by us, i.e., AMMLSEP, AMPH-MML, AMPH-LHD, and C-AMPH-LHD.

Table 4 – Methods used in this thesis to recognize words. DL, MPH and MF stands for “Damerau-Levenshtein”, “Minimum Perfect Hashing” and “Mor-Fraenkel”, respectively.

Methods	$p(s t)$ estimate	Mor-Fraenkel	Search
spelling correction			
FastSS	DL error counting	Symmetric	Hashing
Norvig (2009)	DL error probabilities	–	Neighborhood generation
Boytsov (2012)	DL error counting	Symmetric	MPH
Garbe (2012)	DL error counting	Symmetric	Hashing, prunable
eye-typing word recognition			
Filteryedping	Deletion errors	–	Linear heuristic
AMMLSEP	MML error probabilities (MEP)	Asymmetric	Trie
AMPH-MML	MEP	Asymmetric	MPH, prunable
AMPH-LHD	MEP linear approximation	Asymmetric	MPH, prunable
C-AMPH-LHD	MEP linear approximation + context	Asymmetric	MPH, prunable

Asymmetrical Mor-Fraenkel with MPH and a Linear Heuristic Distance metric (AMPH-LHD) Contextual Asymmetrical Mor-Fraenkel with MPH and a Linear Heuristic Distance metric (C-AMPH-LHD)

As observed, our methods are *asymmetric* Mor-Fraenkel variants which search d -residual strings in an index of k -residual strings ($d \geq k$). As in the work by [Norvig \(2009\)](#), our methods use specific edit probabilities to estimate the likelihood $P(s|t)$. Similarly to the reported by [Allison & Yee \(1990\)](#), these probabilities are used to estimate a MML Optimum Alignment. Unlike previous approaches, the estimates combine general and specific domain knowledge

about errors. Our first method, *Asymmetrical Mor-Fraenkel with MML alignment and Specific Edit Probabilities* (AMMLSEP) (cf. Chapter 3) uses a Trie structure to retrieve candidate words, as suggested by Boytsov (2011). To improve the time and space performance of AMMLSEP, we proposed AMPH-MML (Asymmetrical Mor-Fraenkel with Minimal Perfect Hashing and MML alignment) and AMPH-LHD (Asymmetrical Mor-Fraenkel with Minimal Perfect Hashing and a Linear Heuristic Distance metric) (cf. Chapter 4). AMPH-MML uses a MPH instead of a Trie while AMPH-LHD extends it by adopting a linear estimate of MML and supporting pruning processing with little loss in precision. Finally, we improved a few aspects of the previous methods and proposed C-AMPH-LHD (Contextual Asymmetrical Mor-Fraenkel with Minimal Perfect Hashing and a Linear Heuristic Distance metric) (cf. Chapter 5). This method uses contextual probabilities to support eye movement variability, better balances the influence of language and noise models, and better explores its mode filter. We present and evaluate AMMLSEP, AMPH-MML, AMPH-LHD, and C-AMPH-LHD in the next chapters. The one with the best results, C-AMPH-LHD, is compared to Filterypedping in a real eye-based typing experiment in Chapter 6.

WORD RECOGNITION AS SPELLING CORRECTION

The literature on spelling correction assumes a low error probability and a small number of errors per word. These assumptions do not hold for eye typing input systems, where insertions are more frequent than other errors. In this chapter, we address these problems and propose a spelling correction method for eye typing systems exploring the combination of general and specific domain knowledge about errors and an MML optimum alignment.

3.1 Eye typing Edit Probabilities

In Chapter 2, we presented causes of errors frequently found in other input systems. Most of them can also affect very noisy input systems, as eye-typing ones. Since spurious matches can be relevant and we can capture these factors in large collections of common errors, we adopted an approach based on the methods proposed by [Norvig \(2009\)](#) and [Allison, Wallace & Yee \(1992\)](#). Thus, we modify Eq. 2.2 by estimating costs as:¹

$$\delta_{s[i] \rightarrow \epsilon} = P_d P_E(t_{[j]} s[i] \rightarrow t_{[j]}) \quad (3.1)$$

$$\delta_{\epsilon \rightarrow t_{[j]}} = P_i P_E(t_{[j-1]} \rightarrow t_{[j-1]} t_{[j]}) \quad (3.2)$$

$$\delta_{s[i] \rightarrow s[i]} = P_m P_E(s[i]) \quad (3.3)$$

$$\delta_{s[i] \rightarrow t_{[j]}} = P_s P_E(s[i] \rightarrow t_{[j]}) \quad (3.4)$$

$$\delta_{s_{[i-1:i]} \rightarrow s_{[i:i-1]}} = P_t P_E(s_{[i-1:i]} \rightarrow s_{[i:i-1]}) \quad (3.5)$$

¹ Note that, as [Allison, Wallace & Yee \(1992\)](#), we adopt an interpretation based on Information Theory. However, for clarity of exposition, we omit logarithms in the equations.

where P_d , P_i , P_m , P_s , and P_l are the probabilities of each kind of operation and $P_E(e)$ is the probability of each specific edit operation e . P_d , P_i , P_m , P_s , and P_l can be derived from a small training sample for the target domain. Regarding $P_E(e)$, in cases of insertion and deletion, we use notations $P_E(t_{[j]}s_{[i]} \rightarrow t_{[j]})$ and $P_E(t_{[j-1]} \rightarrow t_{[j-1]}t_{[j]})$ to make it clear that the context is considered, such as [Norvig \(2009\)](#) does. Thus, for instance, $P_E(sd \rightarrow s) > P_E(pd \rightarrow p)$, if the deletion of d is more probable after letter s than after letter p . We assume that for deletions/insertions at the first position, the (previous) context is given by the start of the string.

According to [Norvig \(2009\)](#), e represents a specific edit operation and $P_E(e)$ should be estimated from a corpus of common errors, if this (large enough) corpus is available. If this is not the case for a given scenario S , such as eye-typing systems, then $P_E(e)$ could be obtained by combining (i) estimates extracted from a general corpus with (ii) estimates derived from particular characteristics of the noise observed in S . This way, using a simple linear combination $P_E(e)$ could be given by Eq. 3.6:

$$P_E(e) = \alpha P_C(e) + (1 - \alpha) P_N(e) \quad (3.6)$$

where $P_C(e)$ is the probability associated with e extracted from a corpus of common errors, $P_N(e)$ is the probability associated with e estimated from noise characteristic of scenario S , and α is a factor used to balance the importance of each source of evidence. Also, as the corpus of errors has no estimates for matching, setting $\alpha = 1$, $P_E(e)$ corresponds to the same estimates used by [Norvig \(2009\)](#). For $\alpha < 1$, the estimates for the eye typing scenario become relevant and are discussed in the following paragraphs.

From the previous observations, we note that, given a target character a , the probability that a character b substitutes a or is inserted before/after a is inversely proportional to the distance between a and b . Suppose the interface is defined by sets of points (x, y) , $x \in X$ and $y \in Y$, the center of key a is the most probable point targeted by the eye when looking at a and the error detected by the system is dominated by the Gaussian errors previously described, represented by a covariance matrix Σ . Then, the probability of an arbitrary point (x_i, y_i) , $x_i \in X$ and $y_i \in Y$, mistaken as a point of a can be estimated by the bi-variate distribution given by $\mathcal{N}_2(\mu, \Sigma)$, where mean vector $\mu = (0, 0)$ corresponds to the center of a . If we assume that the dominant error can be represented by an overall standard deviation σ for both X and Y , and there is no correlation between X and Y , then, $\Sigma = \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix}$. Thus, the probability of a point (x, y) being detected by the tracker when the user is looking at a is given by Eq. 3.7 and illustrated by Figure 3 for key d :

$$P(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (3.7)$$

We can now estimate probabilities $P_N(t_{[j]}s_{[i]} \rightarrow t_{[j]})$ and $P_N(s_{[i]} \rightarrow t_{[j]})$ by sampling points around $s_{[i]}$ using Eq. 3.7 and the layout of the visual keyboard. Regarding matches, given two

Figure 3 – Probability that point (x,y) belongs to key d in QWERTY keyboard scenario where dominant error source is the eye tracker with error radius of about 1° ($2\sigma^2 = 1^\circ$).

Source: (HANADA *et al.*, 2016)

strings s and t , the probability of finding a spurious match between two arbitrary characters of s and t is larger for more frequent characters. Thus, we estimate $P_N(s_{[i]} \rightarrow s_{[i]})$ as the inverse of the popularity of $s_{[i]}$ in the language. Finally, we consider that the remaining probabilities are the same as observed in a general corpus.

3.2 Fast Approximate Searching Method for Eye Typing

Lookup methods for approximate searching become unfeasible with high k values because they have to deal with many candidates. Take, for instance, the word presence. If we consider all unique candidates which can be derived by up to k edit operations, we get 442 for $k = 1$, about 90.8k for $k = 2$, and 11.7M for $k = 3$. This exponential growing is mainly due the high number of possibilities generated by the insertion and substitution of $|\Sigma|$ letters at each position of the input pattern. If we consider only deletions, these numbers are quite small: 9 candidates for $k = 1$, 36 for $k = 2$, and 86 for $k = 3$. In fact, Mor-Fraenkel approach takes advantage of this fact by processing only residual strings derived from input string s and dictionary words t . As deletions in t are insertions in s and substitutions and transpositions are combinations of deletions and insertions, all basic operations are covered. Therefore, a lookup algorithm could be still affordable if k remains restricted to a small value but freeing deletion operations from this limitation. Deletions, in its turn, could be limited up to d occurrences, $d \geq k$. This is possible in our scenario since, as seen in Table 2, the most likely errors are insertions. As such, the input pattern will be transformed into correct candidates most likely by applying deletions. An algorithm which would search for candidates with up to d deletions and up to k other operations, could find promising candidates even with a small k value.

Thus, by keeping k small, we can devise a Mor-Fraenkel approach for searching candidates with large distances if such distances are limited to k operations other than deletions. The idea is to store a deletion-only k -neighborhood for all words t in the dictionary. During search, a deletion-only d -neighborhood is built for search pattern s . If a d -residual string, derived from s , matches a word t , then $ED(s,t) \leq d$ deletions; if a k -residual string derived from t matches s ,

then $ED(s, t) \leq k$ insertions; and if a d -residual string derived from s matches a k -residual string derived from t , then $ED(s, t) \leq d$ edit operations with at least $d - k$ deletions. In what follows, we describe this idea in more details.

During indexing, the algorithm iterates over all dictionary strings t , generating its k -deletion neighborhoods, i.e, all k -residual strings t' obtained from t by applying up to k deletions. For each t' we store the deletion lists (t', C^t, P^t) , where $C^t = (t_{[p_1]}, t_{[p_2]}, \dots, t_{[p_\ell]})$ are the characters deleted from t at positions $P^t = (p_1, p_2, \dots, p_\ell)$, $\ell \leq k$. Note that t is easily reconstructed using information in the deletion list. These lists constitute the deletion index which will be used to search dictionary words and its residuals.

During lookup, given an input pattern s , we generate its d -deletion neighborhood. Thus, we search for each d -residual string s' , obtained by deleting characters from s at positions $P^s = (p_1, p_2, \dots, p_m)$, $m \leq d$. The following outcomes are possible:

- s' matches t' with $|P^s| = |P^t| = 0$: s was found as a dictionary word, thus $P(s|t) = \prod_{i=1}^{|t|} P_E(t_{[i]})$ in Eq. 2.1.

- s' matches t' with $|P^s| = 0$ and $|P^t| > 0$: s matched a k -residual of a dictionary word, thus $ED(s, t) \leq k$ insertions and $P(s|t)$ is given by:

$$P(s|t) = \prod_{i \notin P^t} P_E(t_{[i]}) \prod_{i \in P^t} P_E(t_{[i-1]} \rightarrow t_{[i-1]}t_{[i]})$$

- s' matches t' with $|P^s| > 0$ and $|P^t| = 0$: a d -residual of s matched a dictionary word, thus $ED(s, t) \leq d$ deletions and $P(s|t)$ is given by:

$$P(s|t) = \prod_{i \notin P^s} P_E(s_{[i]}) \prod_{i \in P^s} P_E(s_{[i-1]}s_{[i]} \rightarrow s_{[i-1]})$$

- s' matches t' with $|P^s| > 0$ and $|P^t| > 0$: This is a match between k - and d -residual strings with $ED(s, t) \leq d + k$ operations. Since it is not clear which operations are in the optimum alignment, $P(s|t)$ has to be calculated by using Eqs. 2.2 to 3.6.

Supposing $d \geq k$, the search complexity (the upper bound for the average case, based on Boytsov (2011)) of our approach is given by $O(|s|^{d+1} + \frac{|s|^{d+k}L(|s|, d)}{|\Sigma|^{|s|-d}})$, where $N(i)$ is the number of dictionary strings containing i characters and $L(|s|, d) = \max_{|i-|s|| \leq d} N(i)$. The index overhead is about $O(\mu^{k+1}N)$, for N dictionary entries and maximum dictionary word length μ . We represent the deletion index using a full trie, so that the deletion neighborhoods are obtained by traversing the trie during search. We adopt a trie because it was slightly better than hashing implementations for all natural language datasets tested by Boytsov (2011).

To summarize, our proposed method combines general and specific domain knowledge about errors using an MML optimum alignment and specific edit probabilities. Its lookup component consists of an asymmetric Mor-Fraenkel approach where d - and k -residual strings

are matched, with d likely larger than k . From now on, we refer to this method as AMMLSEP — Asymmetrical Mor-Fraenkel with MML alignment and Specific Edit Probabilities.

3.3 Experiments

3.3.1 Baselines

Our baselines are the Mor-Fraenkel variants FastSS as proposed by Bocek, Hunt & Stiller (2007), and Symspell as presented by Garbe (2012). These algorithms look for candidates with Damerau-Levenshtein edit distance $\leq k$. FastSS adopts a delete index as described in Section 3.2 whereas Symspell stores the k -residuals with pointers to their source words instead of deletion lists.

For our experiments, we use the publicly available Java implementations of FastSS² and Symspell³ with no modifications. Thus, candidates are ranked by these algorithms according to edit distance and word frequency. We also adopt the method proposed by Norvig (2009) as baseline. This is a neighborhood generation method, with memorization of repeated strings using hashing sets. Unlike previous methods, it ranks the candidates according to their edit scripts and specific edit probabilities extracted from an error corpus. As a straightforward recursive implementation of a neighborhood approach, we implemented it in Java, easily extending it to support until d deletions and k other operations using a Trie.

3.3.2 Dataset

To evaluate the methods, we first collected words typed by users using a visual keyboard. To accomplish this, we carried out the transcription experiment described by Pedrosa *et al.* (2015), using their eye typing system, Filteryedping. Unlike the user experiment to be presented in the Chapter 6, this experiment was not for comparisons purpose. Thus, we will briefly present the procedures for clarification.

In this experiment, six participants typed random phrases shown to them on the screen. Each subject used the system in three sessions and each session comprised a block of warm-up to familiarize the participants with the interface (not recorded) and four 5-minutes blocks of eye typing. The experiment was carried out on the premises of the Institute of Mathematics and Computer Sciences (ICMC) at the University of São Paulo (USP). Although 5 participants were Brazilian and had the Portuguese as their most fluent language, all of them spent more than 9 months in other countries using English as a communication language. The eye gazing on the keyboard was captured by an eye tracker device and translated into a stream of characters.⁴ This

² <<http://fastss.csg.uzh.ch/>>

³ <<https://github.com/gpranav88/symspell>>

⁴ In this experiment we used a low cost 30 Hz device called EyeTribe.

Table 5 – Edit operation probabilities estimated from word pairs typed by six participants. Confidence intervals (Cf. Int.) are given for a 95% confidence level.

User	#pairs	P_i	P_d	P_s	P_t	P_m
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
Average		0.023	0.289	0.059	0.001	0.628
Cf. Int. ($\pm$)		0.014	0.094	0.017	0.000	0.084

Source: (HANADA *et al.*, 2016)

stream was then filtered out to reduce noise (cf. Table 1).⁵ This experiment allowed us extract 3,073 pairs of strings s and t , s being the output of a mode filter, and t being the word the user was typing.

3.3.3 Evaluation Methodology

Using the pairs extracted from the eye-typing experiment, we can estimate probabilities P_i , P_d , P_s , P_t , and P_m by calculating the optimal edit scripts according to Damerau-Levenshtein distance. Table 5 shows these estimates for each user. To properly assess how AMMLSEP generalizes for each user, we use *6-fold cross validation* (as detailed by Witten & Frank (2011)). We partition the pairs into 6 subsamples, one for each user. From the 6 subsamples, a single one is used as test set. We used the remaining 5 to estimate P_i , P_d , P_s , P_t , and P_m . The process is repeated 6 times, with each one of the 6 users being used once as the test set. The partitions used for training and test were the same in all experiments. As the reported results were averaged over all test pairs, we also provide confidence intervals considering a 95% confidence level. We estimate probability $P(t)$ in Eq. 2.1 from word frequencies extracted from public dictionaries. Similarly, we extracted specific edit probabilities from the same corpus adopted by Norvig (2009).⁶

We evaluate the methods according to space, time, and ranking quality. We measure the space by reporting the size of the indexes kept in memory during test. Regarding time efficiency, all experiments were run on a 3.47Ghz 6-core i7 machine with 16GB of RAM. We used Linux OS, kernel 3.13.x, and Java 1.8, JVM HotSpot 64-Bit Server. To take time measurements, we followed the same procedures adopted by Boytsov (2011): (i) inessential background processes were stopped during test time and (ii) each experiment was performed twice with the shortest obtained time being reported so that we minimize the possibility of competition with OS daemons.

⁵ Besides the Kumar smoothing algorithm (cf. Section 2.6) already provided by Filterypedping, we used a mode filter (PETROU; PETROU, 2010) with window size of 10 samples of 33 ms. All characters sampled less than 5 times in the window are discarded every 5 time steps.

⁶ This information is freely available in <<http://norvig.com/ngrams>>

The final performance of the method is given by the mean reciprocal rank (MRR) over all test pairs. This metric exponentially awards hits near the top of the ranking. This is particularly appropriate for our work given the small screen real estate for displaying candidates and the high cost of context changes in eye typing. Thus, for each input string s , we just recover the top-18 candidates when calculating MRR.⁷

3.3.4 Results

To calculate how likely the alignment between input string s and dictionary word t is, AMMLSEP combines information derived from a general corpus and from the estimates related to eye typing, according to Eq. 3.6. In this equation, factor α is used to leverage the two probability sources, varying from $\alpha = 1$ (probabilities are estimated only from the general corpus) to $\alpha = 0$ (they are estimated based only on eye typing error information). Figure 4 shows how α impacts on MRR performance for both AMMLSEP and Norvig, considering dictionaries with 70k, 140k, and 280k words.

Figure 4 – $\alpha \times$ MRR for AMMLSEP, $k = 2$, $d = 10$.

Source: (HANADA *et al.*, 2016)

Figure 5 – $d \times$ MRR and time, for AMMLSEP, $k = 2$, $\alpha = 0.7$.

Source: (HANADA *et al.*, 2016)

As expected, the best results were obtained while combining the estimates derived from the noise model and the general corpus. Best values for α were 0.7 for AMMLSEP and 0.9 for Norvig disregarding the dictionary size. In general, AMMLSEP provided best results when compared to Norvig, even when considering corpus estimates taken in isolation ($\alpha = 1$). This gain was due to the use of matches (Eq. 3.3) and the fact that the algorithm in Section 3.2 finds the *best* solutions for all pairs (s, t) for which $ED(s, t) \leq d + k$ edit operations. Note, however, that the best solution is not limited to $d + k$ edit operations.

Other important parameter for AMMLSEP is the number of deletions (d) to be considered during the neighborhood generation at lookup time. A large d allows the correction of more insertion errors, the commonest error observed on eye typing. However, the larger d is, the larger

⁷ As about six word candidates are shown in each turn in our virtual keyboard, this comprises 3 screen changes.

is the neighborhood and, consequently, the search time. This trade-off is illustrated by Figure 5, which shows MRR and time associated with d , varying from 1 to 14. MRR and time figures were obtained for AMMLSEP with $\alpha = 0.7$ and $k = 2$. As we can see, MRR grows fast as d increases from 1 to 6, and after that, MRR increases very slowly. The behavior is similar for all dictionary sizes. Regarding time, cost grows linearly, being proportional to the size of the dictionary. Although not shown here, we have observed the same results for $k = 1$ and $k = 3$. From these results, we conclude that d values from 6 to 10 provide a good trade-off between ranking quality and search time.

Given α and d , we can now compare AMMLSEP with our baselines, Norvig, Symspell, and FastSS. Figures 6 and 7 compare these methods, emphasizing the impact of index size on MRR and on lookup time. In these figures, we use $\alpha = 0.7$ and $d = 10$ for AMMLSEP. Each method is shown as a line with up to three points, one for each dictionary size (70k, 140k and 280k, respectively) for which the method was able to run. Norvig approach is shown using only its original corpus data ($\alpha = 1$) and with its best result in Figure 4 ($\alpha = 0.9$). Symspell and FastSS results are shown for all cases in which the dictionary indices fit in memory.

Figure 6 – Space x MRR (AMMLSEP $\alpha = 0.7$).

Source: (HANADA *et al.*, 2016)

Figure 7 – Space x time (AMMLSEP $\alpha = 0.7$).

Source: (HANADA *et al.*, 2016)

Figure 6 shows that AMMLSEP ($\alpha = 0.7$) outperforms all the other methods, according to MRR. It also shows that AMMLSEP and Norvig are the most robust to varying dictionary sizes. For the other methods, performance sharply degrades as the dictionary size doubles, because many candidate words are found with the same k distance (Symspell and FastSS). The impact of k is less important for AMMLSEP when compared with other methods. Even for $k = 1$, it outperforms the other methods since it supports up to d deletion operations. Norvig's approach is the only method competitive in ranking quality. Even with the highest possible k values, Symspell and FastSS are not able to achieve performance similar to Norvig and AMMLSEP, because they adopt a very simple estimate strategy based on Damerau-Levenshtein distance and allow few deletions. To be competitive, they have to resort to larger k values, but the index size quickly becomes prohibitive.

In Figure 7, we observe that the large indices used by Mor-Fraenkel methods pay off. These methods are much faster than the neighborhood approach of Norvig. Among the variants of Mor-Fraenkel, AMMLSEP is generally slower than FastSS and Symspell because of (i) the large number of candidates to lookup, generated in searching d -neighborhood and (ii) the adopted alignment estimate, which is more complex. In spite of that, AMMLSEP reaches top ranking performance even for $k = 1$. In fact, by comparing AMMLSEP ($k = 1$, $\alpha = 0.7$, $d = 10$) with the best performance obtained by Norvig ($k = 2$, $\alpha = 0.9$, $d = 10$) in the 140k dictionary, we observe it is 28 times faster to provide a ranking 4% better. Similarly, compared to Symspell ($k = 4$) in the same dictionary, it delivers a ranking 19% better, and twice as fast. When compared to the best overall result provided by FastSS ($k = 3$, 70k dictionary), it was 24% better in MRR, although nine times slower.

To better understand the AMMLSEP robustness to noise and its ranking performance, Figures 8 and 9 show the impact of the string pattern size on MRR and on lookup time. Figure 8 also includes the input string size distribution. Although the largest string in our test set has 35 letters, these figures include only sizes for which at least 15 strings were found (sizes 1 to 19). Since FastSS and Symspell present very similar performance, we report results only for Symspell. AMMLSEP and Norvig use the same setting of the previous experiment. Due to space restrictions, only results for the 140k dictionary were reported.

As seen in Figure 8, AMMLSEP and Norvig approaches significantly outperform Symspell for words with more than 9 characters. This is due to the limited coverage of Symspell with $k = 4$. For larger words, it is more likely that the number of errors (and, in particular, incorrect insertions) are greater than k . We also note that AMMLSEP significantly outperforms Symspell for words with size from 3 to 6 characters (the most frequent sizes) due to the adoption of the MML alignment (for 3-characters words, it is even better than Norvig). As a consequence, this gain is larger for larger dictionaries due to the increasing noise resulting from a larger number of words.

Figure 8 – Size x MRR in 140k dictionary.

Source: (HANADA *et al.*, 2016)

Figure 9 – Size x time (AMMLSEP $\alpha = 0.7$, $k = 2$, $d = 10$).

Source: (HANADA *et al.*, 2016)

Table 6 – Proportion of the 3,073 pairs in which correct word was placed at top position and rank slots 1-6, 7-12, and 13-18.

Method	Top-1	1-6	7-12	13-18
70k dictionary				
AMMLSEP	64.82	84.18	87.63	89.26
Norvig	60.39	82.90	86.75	88.80
Symspell $k = 4$	54.12	75.59	78.59	80.18
FastSS $k = 3$	50.34	68.76	71.33	77.70
140k dictionary				
AMMLSEP	63.65	83.82	87.63	89.32
Norvig	59.22	82.85	86.91	89.03
Symspell $k = 4$	50.54	73.84	77.81	79.30
FastSS $k = 2$	40.25	56.88	59.39	59.91
280k dictionary				
AMMLSEP	62.47	83.20	87.30	88.83
Norvig	58.15	82.39	86.56	88.54
Symspell $k = 3$	42.01	66.25	69.70	71.59
FastSS $k = 2$	36.05	56.00	58.44	59.65

Source: (HANADA *et al.*, 2016)

Figure 9 shows AMMLSEP average lookup time according to the size of the input string pattern s . As expected, the answering time is higher for string sizes which will lead to more dictionary candidates, which also depends on the value of d and the size of the dictionary. For $d = 10$, string sizes 9 to 14 generate many candidates, as illustrated in Figure 9. More candidates imply more pairs (s, t) such that the match between d -residuals of s and k -residuals of t is more likely. As consequence, $P(s|t)$ has to be calculated using the dynamic programming approach (cf. Eq. 2.2) whose cost, for each pair, is proportional to $O(|s| |t|)$.

Finally, for our particular scenario, screen space is a scarce resource and context changes are very expensive since they imply additional (tiring) tasks for the eyes. In such situations, even if the method is not able to rank the correct word at the top position, it should avoid placing the word at a very low rank, which would oblige the user to switch many screens. Table 6 illustrates the ability of AMMLSEP ($\alpha = 0.7$, $k = 2$, $d = 10$) to avoid excessive screen changes, by providing more information about how it ranked candidates for our test set of 3,073 pairs of strings. The table reports the proportion of pairs which were ranked at the top position and at positions 1 to 6, 7 to 12, and 13 to 18. It also shows more details on the best rankings obtained with methods Norvig ($\alpha = 0.9$, $k = 2$, $d = 10$), Symspell, and FastSS.

As we can see, by using AMMLSEP and Norvig, the user will switch screens less frequently than with Symspell and FastSS. We also note that AMMLSEP is the most precise at the top position, and is the one, which obtained better recall. Norvig's approach achieved very close results, specially for the last positions of the rank.

3.4 Final considerations

In this chapter we have presented a spelling correction method for eye typing systems. It combines general and specific domain knowledge about errors using a minimum message length optimum alignment and specific edit probabilities. Its lookup component consists of an asymmetric Mor-Fraenkel approach where d - and k -residual strings are matched, with d likely larger than k . In tests using a real eye typing log data, our method was able to outperform three state-of-the-art baselines, by using specific edit probabilities estimated by combining information from a general corpus and the eye typing domain. This method was also more robust to variations of the dictionary size. Its high precision also contributed to minimize the need for context changes in an eye typing interface. These results were reported in the paper “Effective Spelling Correction for Eye-based Typing using domain-specific Information about Error Distribution” ([HANADA et al., 2016](#)) published in the Proceedings of the 2016 ACM Conference on Information and Knowledge Management – CIKM 2016.

FAST WORD RECOGNITION IN NOISY SCENARIOS

Most of our effort in previous chapter was focused on the effectiveness of AMMLSEP as a word recognizer. Thus, we extended an MML approach with specific edit probabilities towards improving ranking quality. Although AMMLSEP presents a reasonable response time, it is slower than simple Damerau-Levenshtein based approaches. In this chapter, we will improve AMMLSEP efficiency not only to provide good suggestions at adequate response time, but also to ensure that the response time is never greater than a certain predetermined time limit. In this way, we expect to provide a method which can be a viable component of a real user interface system. To accomplish this we propose two prunable methods where Mor-Fraenkel indices are searched using minimal perfect hashing. The first one still estimates $P(s|t)$ using Eq. 2.2 while the second one uses a novel linear heuristic.

4.1 Fast Retrieval with Perfect Hashing

The main advantage of using a Trie to store residual strings, as we did in Chapter 3, is to not waste processing time searching for pattern residuals that have not been seen during indexing. In a scenario of natural language input with a lot of noise, this feature is important since many candidates might be filtered out by the Trie before performing the costly quadratic MML evaluation. However, given a pattern residual s' , the Trie search cost is proportional to its size, i.e., $O(|s'|)$. Further, the indexing of residuals and Mor-Fraenkel deletion lists using a Trie requires a lot of memory. An alternative to cope with these issues is to adopt a perfect hashing (as Botelho, Pagh & Ziviani (2007)) as a more efficient search structure, as suggested by Boytsov (2012).

The idea is to enumerate all dictionary residual strings t' using a minimal perfect hash function $f(\cdot)$. As $f(\cdot)$ maps m strings to integers ranging from 1 to m with no collisions, search cost is $O(1)$. Thus, the indexer iterates over all the dictionary strings t to generate their k -deletion

neighborhoods. As result, each k -residual t' is associated with a deletion list¹. All the lists generated are stored in a deletion index using the values of the perfect hash function as keys. During search time, we recursively generate all possible d -residual and their respectively deletion lists. An important problem of this strategy, when compared to triple generation based on a Trie, is the necessity to enumerate all possible residuals before evaluating them, which can be prohibitive for very large search patterns and a high number of deletions d . In a Trie, only residuals that have been seen during indexing can be traversed. Thus, if a string s' was not observed at index time, it will not be searched, nor any other string for which s' is a prefix, which drastically reduces the search space.

A simple way to cope with this issue is to restrict the neighborhood generation to a certain time limit. However, to safely apply time limits, it is desirable to first generate the most promising residuals. In other words, given the search pattern $s = \text{minds}$ and a 3-neighborhood, we would prefer to firstly match the residual $s' = \text{minds}$, obtained with no deletions, than the residuals mind or in , obtained with 1 and 3 deletions, respectively. Thus, we decided to generate the η -neighborhood recursively in a start to finish depth-first way, according to Algorithm 1.

In Algorithm 1, the generation of the neighborhood is performed by the recursive function `Get- $\eta$ -neigh-r`. Given a string w , all their residuals w' are generated only a single time (lines 19-21), along with their corresponding sets of deleted characters and positions, $C^{w'}$ and $P^{w'}$ (line 22). The algorithm is then called recursively to generate the residuals of w' (lines 23-29). Note, in line 22, that deletion index I is created such that I_d refers to all residuals generated by the application of exactly d deletions. Thus, to define the set of candidate words within a time limit τ , we iterate through the pattern residuals s' , ordered according to many deletions were used to generate them, as shown in Algorithm 2.

Algorithm 2 implements an asymmetrical MF approach combined with a perfect hashing. Thus, given a search pattern s , it is able to rank a set of candidates $t_i \in \mathcal{D}$, such that $\mathcal{E}_{DEL}(s, t_i) \leq d$ and $\mathcal{E}_{IST}(s, t_i) \leq k$ with d probably larger than k , i. e. we first generate the d -deletion neighborhood of s (line 6). $\mathcal{N}$ provides the deletion index of all d -residual strings s' , ordered by the number of deletions applied. We iterate through them (lines 8-20), retrieving from the dictionary all triples $(f(t'), C^t, P^t)$ such that $f(s') = f(t')$ (line 10). If this process takes more than τ milliseconds, only the suggestions calculated so far are returned (lines 13-14).

Note that $f(s') = f(t')$ implies that $s' = t'$ only if s' was part of the set of residual dictionary strings computed at index time, according to the definition of a perfect hash function. If this is not the case, $s' \neq t'$ and the pair (s', t') is a false positive. To detect that, it is necessary to reconstruct the string t , associated with $f(s')$, by inserting into s' , at positions $P^{t'}$, the characters in $C^{t'}$. If t is not found in $\mathcal{D}$, it is recognized as a false positive and none of the triples $(f(s'), C^t, P^t)$, $f(s') = f(t')$, belongs to the dictionary (lines 15-17). Although expensive, we have to perform such reconstruction only once for every residual dictionary string t' .

¹ A detailed explanation of deletion lists is given in Section 3.2

Algorithm 1: Get η -neighborhood

```

1 Function Get- $\eta$ -neighborhood( $w, \eta$ )
2    $w$ : target word for  $\eta$ -neighborhood generation;
3    $\eta$ : the max number of deletion to perform;
4    $p$ : starting position on  $w$ ;
5   for  $d = 0$  to  $\eta$  do
6     | Create empty deletion index  $I_d$ ;
7     add  $w$  to  $I_0$ ;                                /*  $w$ : single  $w$  residual with 0 dels */
8      $I_0[w] \leftarrow []$ ;                          /* so, its deletion list is empty */
9      $C^w \leftarrow []$ ;                             /* deleted chars in current del list */
10     $P^w \leftarrow []$ ;                             /* ... and its positions */
11     $O^w \leftarrow (1, \dots, |w|)$ ;                /* original positions of  $w$  chars */
12     $d \leftarrow 0$ ;                                /* #deletions already performed */
13    if  $\eta > 0$  then
14      | return Get- $\eta$ -neigh-r( $w, O, I, C^w, P^w, d, \eta, 0$ );
15    else
16      | return  $I$ ;
17 Function Get- $\eta$ -neigh-r( $w, O, I, C^w, P^w, d, \eta, p$ )
18    $d \leftarrow d + 1$ ;
19   if  $|w| > 1$  then
20     | for  $i = p$  to  $|w|$  do
21       |  $w' \leftarrow w_{[0:i]} + w_{[i+1:]}$ ;
22       | add  $(C^w + w_{[i]}, P^w + O_i)$  to  $I_d[w']$ ;
23       | if  $d < \eta$  then
24         |  $k \leftarrow 1$ ;
25         | for  $j = 1$  to  $|w|$  do
26           | if  $i \neq j$  then
27             |  $newO_k \leftarrow O_j$ ;
28             |  $k++$ ;
29       | Get- $\eta$ -neigh-r( $w', newO, I, C^w + w_{[i]}, P^w + P[i], d, \eta, i$ );
30   return  $I$ ;

```

If $f(s') = f(t')$ and this is not a false positive case, t can be reconstructed as previously described. Given s and t and if t has not already been included in the suggestion list (lines 18-19), it is possible to estimate the distance between them using the MML algorithm (line 20).

In our implementation, the perfect hash functions are computed using `minperf`², a library written in Java able to concurrently generate minimum perfect hash functions that need less than 1.58 bits per key. From now on, we refer to our asymmetrical Mor-Fraenkel method with minimum perfect hashing and MML metric as AMPH-MML.

² <<https://github.com/thomasmueller/minperf>>

Algorithm 2: Lookup residuals of search pattern

```

1 Function LookupResiduals( $s, I, \tau$ )
2    $s$ : search pattern;
3    $I$ : deletion index corresponding to all the dictionary word residuals  $t'$ ;
4    $\tau$ : lookup timeout;
5    $S \leftarrow []$ ;                                /* list of suggestions */
6    $\mathcal{N} \leftarrow \text{Get-}\eta\text{-neighborhood}(s, d, \tau)$ ;
7   for  $dels$  from 0 to  $d + 1$  do
8     for each  $s'$  in  $\mathcal{N}_{dels}$  do
9        $\mathcal{L}_{s'} \leftarrow \mathcal{N}_{dels}[s']$ ;           /* del lists for residuals of  $s$  */
10       $\mathcal{L}_{t'} \leftarrow I[f(s')]$ ;           /* del lists for  $t', f(s') = f(t')$  */
11      for each  $(f(s'), C^{s'}, P^{s'})$  in  $\mathcal{L}_{s'}$  do
12        for each  $(f(t'), C^{t'}, P^{t'})$  in  $\mathcal{L}_{t'}$  do
13          if current running time  $> \tau$  then
14            return  $S$ ;
15           $t \leftarrow \text{reconstruct}(s', C^{t'}, P^{t'})$ ;
16          if first time in  $\mathcal{L}_{t'}$  and  $t \notin \mathcal{D}$  then
17            break;
18          if  $t \in S$  then
19            break;
20           $S \leftarrow \text{Evaluate}(s, t, S)$ ;
21  return  $S$ ;
22 Function Evaluate( $s, t, S$ )
23   score  $\leftarrow \text{MML}(s, t)$ ;
24   add  $(t, \text{score})$  to  $S$ ;
25  return  $S$ ;

```

4.1.1 Complexity Analysis

The neighborhood generation algorithm (Algorithm 1) generates all possible combinations of d deletions on a string s recursively, and thus generates $\binom{|s|}{d}$ residual strings in the η -neighborhood. In the worst case, where all those residuals match with the entries in the index with the largest number of residual tuples $\mathcal{L}_{max}$, the upper bound of our approach is $O(\binom{|s|}{d} |\mathcal{L}_{max}| |s|^2)$, where $|s|^2$ is the cost of the MML score evaluation of a s', t' candidate pair.

The index overhead, generated mainly by the storage of the (t', C^t, P^t) triples, has upper bound for the worst case of $O(\binom{\mu}{k} * N * \mu)$, where N is the number of dictionary strings and μ is the size of the largest string in the vocabulary, with $\binom{\mu}{k}$ being the number of elements in the k -neighborhood of a μ -sized string, and $N * \mu$ being proportional to the cost of storing reconstruction triples.

4.2 Linear Distance Heuristic

As observed in Algorithm 2, the distance estimate between s and t is carried out by a quadratic algorithm that does not take advantage of matches already known in the deletion index. We now propose a linear heuristic that explores such information.

If string s can be transformed into t after d deletions in s and k deletions in t (d -residual s' is equal to k -residual t'), s and t can be rewritten as $s = s_0 s'_{[0]} \dots s_{k-1} s'_{[k-1]} s_k$ and $t = t_0 t'_{[0]} \dots t_{k-1} t'_{[k-1]} t_k$, respectively. Since $s' = t'$, $t = t_0 s'_{[0]} \dots t_{k-1} s'_{[k-1]} t_k$. Thus, transforming s into t corresponds to perform transformations $s'_{[0]} \rightarrow s'_{[0]}, \dots, s'_{[k-1]} \rightarrow s'_{[k-1]}$, $s_0 \rightarrow t_0, \dots, s_{k-1} \rightarrow t_{k-1}$, and $s_k \rightarrow t_k$. As a running example, suppose $s = \text{bacon}$ and $t = \text{aunt}$, with $s' = t' = \text{an}$. Transforming s into t corresponds to $b \rightarrow \epsilon$, $a \rightarrow a$, $co \rightarrow u$, $n \rightarrow a$, and $\epsilon \rightarrow t$.

A transformation $s_i \rightarrow t_i$ can be rewritten as either $w \rightarrow \epsilon$ (e.g., $b \rightarrow \epsilon$), $\epsilon \rightarrow w$ (e.g., $\epsilon \rightarrow t$), or $v \rightarrow w$ (e.g., $co \rightarrow u$), where v and w are strings such that $v \neq \epsilon$ and $w \neq \epsilon$. Transformation $w \rightarrow \epsilon$ corresponds to the deletion of characters $w_{[0]}, \dots, w_{[|w|-1]}$ from s while $\epsilon \rightarrow w$, to the insertion of $w_{[0]}, \dots, w_{[|w|-1]}$ into s . Unlike these previous cases, $v \rightarrow w$ may have more than a single edit script because different combinations of operations can yield the same result. For instance, $co \rightarrow u$ is obtained by either $c \rightarrow u, o \rightarrow \epsilon$ or $c \rightarrow \epsilon, o \rightarrow u$. Further, substitutions and transpositions can be rewritten as combinations of the other operations (insertions, deletions, and matches). As consequence, to find the most probable edit script, all the alternatives should be inspected.

However, assume that transpositions are not allowed and substitutions are more likely than its corresponding pairs of deletions and insertions. If $|w| = |v|$, the transformation $v \rightarrow w = v_{[0]}, \dots, v_{[|v|-1]} \rightarrow w_{[0]}, \dots, w_{[|w|-1]}$ corresponds to a succession of matches or substitutions, depending on $v_{[i]} = w_{[i]}$ (match) or $v_{[i]} \neq w_{[i]}$ (substitution). Note that the case where $|w| \neq |v|$ (e.g., $co \rightarrow u$) is still ambiguous. A simple heuristic to deal with this case is to align chars at excess with ϵ . Thus, for $|v| > |w|$, $v_{[i]} \rightarrow w_{[i]}$ corresponds to a match or substitution if $i < |w|$, or a deletion if $i \geq |w|$. Similarly, for $|v| < |w|$, $v_{[i]} \rightarrow w_{[i]}$ corresponds to an insertion if $i \geq |v|$. As consequence, $co \rightarrow u$ will be interpreted as $c \rightarrow u, o \rightarrow \epsilon$. While this simple heuristic can handle non-optimal solutions, the loss in precision is acceptable, as we will observe in our empirical evaluation.

Given the previous observations and assuming that $(s_{[i]}, t_{[j]})$ is the current pair of characters to be aligned, one of the following situations is possible:

- $i \notin P^s \wedge j \notin P^t$: edit operation is a match of $s_{[i]}$ and next pair to inspect is $(s_{[i+1]}, t_{[j+1]})$.
- $i \in P^s \wedge j \in P^t$: edit operation is a substitution of $s_{[i]}$ by $t_{[j]}$ if $s_{[i]} \neq t_{[j]}$, otherwise is a match of $s_{[i]}$. The next pair to inspect is $(s_{[i+1]}, t_{[j+1]})$.
- $i \notin P^s \wedge j \in P^t$: edit operation is insertion of $t_{[j]}$ and next pair to inspect is $(s_{[i]}, t_{[j+1]})$.
- $i \in P^s \wedge j \notin P^t$: edit operation is deletion of $s_{[i]}$ and next pair to inspect is $(s_{[i+1]}, t_{[j]})$.

A simple way to allow transpositions is by checking if the current operation is not part of a sequence of operations equivalent to a transposition. If this is the case, the sequence of operations is replaced by the corresponding transposition. More specifically, transpositions are equivalent to sequences composed by (i) two substitutions ($ab \rightarrow ba \approx a \rightarrow b, b \rightarrow a$)³, (ii) a deletion, a match and an insertions ($cab \rightarrow cba \approx ca \rightarrow c, b \rightarrow b, b \rightarrow ba$), and (iii) an insertion, a match and a deletion ($cab \rightarrow cba \approx c \rightarrow cb, a \rightarrow a, ab \rightarrow a$). The complete algorithm, supporting transpositions and edit operations with previous context, is shown in Algorithm 3. From now on we refer to it as LHDistance.

Algorithm 3: Linear heuristic

```

1 Function LHDistance( $s, C^t, P^s, P^t$ )
2    $\mathcal{E} \leftarrow []$ ;                                /* List of edit operations */
3    $i \leftarrow j \leftarrow 0$ ;                      /* Current char pos at  $s$  and  $t$  */
4    $m \leftarrow n \leftarrow 0$ ;                      /* Current pos at  $P^s$  and  $C^t/P^t$  */
5    $\rho \leftarrow \langle$ ;                                /* Previous key assumed correct */
6    $\beta_{xy} \leftarrow \text{return } \langle \text{if } x = \langle \text{else } y;$ 
7    $K(e) \leftarrow \text{return}$  kind of edit oper.  $e$  ( $i, d, s, t$ , or  $m$ );
8   while  $i < |s|$  do
9     if  $i = P_m^s \wedge j = P_n^t$  then
10      if  $s[i] = C_n^t$  then
11         $\rho \leftarrow \text{AddMatch}(\mathcal{E}, s[i]);$           /* Match */
12      else
13         $\rho \leftarrow \text{AddSubsOrTrans}(\mathcal{E}, s[i], C_n^t);$  /* Substitution */
14         $i++, j++, m++, n++$ ;
15      else if  $i = P_m^s$  then
16         $\rho \leftarrow \text{AddDelOrTrans}(\mathcal{E}, \rho, s[i], \beta_\rho);$  /* Delete  $s[i]$  */
17         $i++, m++$ ;
18      else if  $j = P_n^t$  then
19         $\rho \leftarrow \text{AddInsOrTrans}(\mathcal{E}, \rho, C_n^t, \beta_\rho);$  /* Insert  $C_n^t (= t[j])$  */
20         $j++, n++$ ;
21      else
22         $\rho \leftarrow \text{AddMatch}(\mathcal{E}, s[i]);$ 
23         $i++, j++$ ;
24   while  $n < |C_n^t|$  do
25      $\rho \leftarrow \text{AddInsOrTrans}(\mathcal{E}, \rho, C_n^t, \beta_\rho);$  /* Insert remaining  $t$  chars */
26      $n++$ ;
27   return  $\sum_i L_{K(\mathcal{E}_i)} + L_E(\mathcal{E}_i);$ 

```

³ In fact, transpositions can also be obtained by other combinations of operations involving substitutions. For example, a substitution S followed by a match M and a deletion D , in which S inserts, before a character c of M , another character that is the same one removed by D after c . In such cases, however, the conversion necessary to include a transposition requires additional operations, which makes less likely an effective reduction of the edit script cost.

Algorithm 4: Functions AddMatch, AddDelOrTrans, AddSubsOrTrans, AddInsOrTrans, and Evaluate.

```

1 Function AddMatch( $\mathcal{E}, c_1$ )
2   | add ( $c_1 \rightarrow c_1$ ) to  $\mathcal{E}$ ;
3   | return  $c_1$ ;
4 Function AddSubsOrTrans( $\mathcal{E}, c_1, c_2$ )
5   | if  $\mathcal{E}_{last} = c_2 \rightarrow c_1$  then
6   |   |  $\mathcal{E}_{last} \leftarrow c_2 c_1 \rightarrow c_1 c_2$ ;
7   | else
8   |   | add ( $c_1 \rightarrow c_2$ ) to  $\mathcal{E}$ ;
9   | return  $c_2$ 
10 Function AddDelOrTrans( $\mathcal{E}, c_1, c_2$ )
11   | if  $\mathcal{E}_{last-1} = k \rightarrow k c_2 \wedge k \neq \langle \wedge c_1 \neq c_2$  then
12   |   |  $\mathcal{E}_{last-1} \leftarrow c_1 c_2 \rightarrow c_2 c_1$ ;
13   |   | delete  $\mathcal{E}_{last}$ ;
14   | else
15   |   | add ( $c_1 c_2 \rightarrow c_1$ ) to  $\mathcal{E}$ ;
16   | return  $c_1$ 
17 Function AddInsOrTrans( $\mathcal{E}, c_1, c_2$ )
18   | if  $\mathcal{E}_{last-1} = k c_2 \rightarrow k \wedge k \neq \langle \wedge c_1 \neq c_2$  then
19   |   |  $\mathcal{E}_{last-1} \leftarrow c_2 c_1 \rightarrow c_1 c_2$ ;
20   |   | delete  $\mathcal{E}_{last}$ ;
21   | else
22   |   | add ( $c_1 \rightarrow c_1 c_2$ ) to  $\mathcal{E}$ ;
23   | return  $c_2$ 
24 Function Evaluate( $s, t, C^t, P^s, P^t, S$ )
25   | score  $\leftarrow$  LHDistance( $s, C^t, P^s, P^t$ );
26   | add ( $t$ , score) to  $S$ ;
27   | return  $S$ ;

```

As seen in Algorithm 3, LHDistance parameters are the input pattern s and its deletion list P^s as well as the deletion list P^t and corresponding character list C^t . Indices i and j indicate the current pair of chars to be aligned. String t is not required as all necessary information is in the deletion index. Variables n and m are used to access the deletion lists. The edit operations are stored in $\mathcal{E}$.

To correctly infer the edit operations, LHDistance keeps track of the context using variable ρ . In particular, ρ saves the previous character, believed to be the correct one. Variable ρ is necessary because it provides context for a possible following deletion or insertion operation. Initial context is always the start of the word ($\langle$). To illustrate param ρ , suppose a user typing vip misinserts o, yielding $s = viop$. To correct the error, o has to be deleted. After such deletion, and following the work by Norvig (2009), the context for the insertion of p is $\rho = i$, resulting

in $i \rightarrow ip$. The resulting probability would be $P(i \rightarrow ip)$, that is, the probability of inserting p after i . Parameter ρ depends on the inferred edit operation. Thus it is defined in functions `AddMatch`, `AddSubsOrTrans`, `AddDelOrTrans`, and `AddInsOrTrans` (cf. Algorithm 4).

All pairs $(s_{[i]}, t_{[j]})$ are aligned from line 9 to 26. Substitutions, deletions and insertions are added to $\mathcal{E}$ only after they had been verified not to be part of a transposition (functions `AddSubsOrTrans`, `AddDelOrTrans`, and `AddInsOrTrans`). After determining all the edit operations, the edit script probability is calculated according to Eq. 2.2 (line 27). Finally, function `Evaluate`, originally defined in Algorithm 2, is modified to consider the full Mor-Fraenkel index information.

The adoption of `LHDistance` instead of the original MML in our approach has the impact of lowering the cost of the evaluation function from $|s|^2$ to $2|s|$. Thus, the upper bound of the complexity in this case is $O(\binom{|s|}{d} |L_{max}| |s|)$.

In this work, we refer to our asymmetrical Mor-Fraenkel method with minimum perfect hashing and linear-heuristic based metric as AMPH-LHD.

4.3 Experiments

4.3.1 Baselines

Since our methods are focused on scenarios characterized by noise, our main baseline is the work reported in Chapter 3, AMMLSEP method. As a fast approximate dictionary approach, we adopt the work reported by Boytsov (2012) which we refer to as *SISAP12*.⁴ *SISAP12* adopts a Mor-Fraenkel (MF) approach with perfect hashing which looks for candidates using Damerau-Levenshtein edit distance $\leq k$. AMMLSEP uses an asymmetrical MF approach with a Trie which looks for candidates with $\mathcal{E}_{DEL} \leq d$ and $\mathcal{E}_{IST} \leq k$. In our experiments, we used the same Java implementation of AMMLSEP reported in Chapter 3. Regarding *SISAP12*, we developed a Java port based on its original C++ code.⁵ Note that the original *SISAP12* implementation compresses the MF index to evaluate its impact on scalability. Since Boytsov uses a very simple compression technique applicable to any MF index, it can also be used by our methods. However, as the original AMMLSEP do not use any index compression, to provide a fair comparison, we disabled compression in *SISAP12*. In *SISAP12*, the candidates are ranked according to edit distance and word frequency, while AMMLSEP ranks candidates according to their edit scripts and specific edit probabilities extracted from an error corpus.

⁴ Although the work presented by Chegrane & Belazzougui (2013) is as efficient as *SISAP12* and has a faster dictionary building algorithm, we prefer the later because it supports transpositions.

⁵ <http://searchivarius.org/personal/jea2011/>

Folds	#pairs	P_i	P_d	P_s	P_t	P_m
EYE-PAIRS dataset						
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
ASWIKI-PAIRS dataset						
1	481	0.052	0.029	0.054	0.016	0.849
2	481	0.052	0.029	0.053	0.016	0.849
3	481	0.051	0.029	0.054	0.015	0.851
4	481	0.052	0.029	0.054	0.016	0.849
5	481	0.051	0.029	0.054	0.016	0.851
6	477	0.051	0.029	0.054	0.015	0.850

Table 7 – Edit operation probabilities

Source: (CRISTO *et al.*, 2017)

4.3.2 Evaluation Methodology

We used the same dataset described in Section 3.3.2 (EYE-PAIRS) as our list of noisy words for recognition. To better observe the impact of noise in the performance of the methods, we also used a dataset of typical misspellings. We refer to this dataset as ASWIKI-PAIRS. It was created by combining datasets Aspell and Wikipedia, provided by Roger Milton⁶. Aspell is a set of 531 misspellings of 450 words used to test the GNU Aspell spellchecker. The Wikipedia dataset is a list of misspellings made by Wikipedia authors, as described by Deorowicz & Ciura (2005), containing 2,455 misspellings of 1,922 words. After removing duplicates, ASWIKI-PAIRS was left with 2,882 pairs of strings s and t (s always different from t).

As our methods use Eq. 2.2 to calculate the similarity of candidates, we need to estimate L_i , L_d , L_s , L_t , and L_m ⁷ for EYE-PAIRS and ASWIKI-PAIRS. For EYE-PAIRS, we used the estimates adopted in Chapter 3. For ASWIKI-PAIRS, we estimated these probabilities from the Damerau-Levenshtein distances of the pairs of strings, using six-fold cross validation (detailed by Witten & Frank (2011)). We have chosen six-fold validation because EYE-PAIRS dataset contains pairs generated by six different users. Thus, we split the pairs in 6 partitions. Regarding EYE-PAIRS, in particular, each partition corresponds to a different user while in ASWIKI-PAIRS the partitioning is random. As a result, probability estimates and all reported evaluation metrics for AMMLSEP, AMPH-MML and AMPH-LHD is obtained as the average of six rounds: at each round, a different partition is used as test set where the remaining partitions are used for training. The probability estimates we obtained is shown in Table 7. Note that character

⁶ <<http://www.dcs.bbk.ac.uk/~ROGER/corpora.html>>

⁷ In this chapter, we call L_i , L_d , L_s , L_t , and L_m the variables corresponding to the negative logarithms of probabilities P_i , P_d , P_s , P_t , and P_m from Eq. 2.2, while $L_E(e)$ is the probability logarithms of specific edit operation.

modifications are more common in the noisy EYE-PAIRS, with insertions more prominent than in ASWIKI-PAIRS.

To estimate the probability logarithms of specific edit operations, $L_E(e)$, we use frequencies of edit operations observed in the general corpus used by Norvig (2009) and the noise specific information reported in Chapter 3. In particular, for EYE-PAIRS, we consider $L_E = -\log(0.3P_C + 0.7P_N)$, where P_C indicates probabilities associated with the general corpus and P_N , probabilities associated with noise. For EYE-ASWIKI, $L_E = -\log P_C$. The datasets with the word frequencies (dictionaries) and frequencies of edit operations were the same described in Chapter 3. The edit operations were mostly based on the Birkbeck spelling error corpus, a collection with 36,133 misspellings of 6,136 words, from the Oxford Text Archive.

We can estimate the contextual component $P(t)$, in Eq. 2.1, with sophisticated n-gram models as in the work by Bassil (2012) and Islam & Inkpen (2009). However, since our focus is on $P(s|t)$, we adopted a simple unigram model with word frequencies extracted from public dictionaries. This model can be easily extended to support n-grams if the additional space necessary to store the n-gram statistics is not a concern.

As in Chapter 3, we compared the proposed methods according to time, space and ranking quality, providing confidence intervals and significance test results, for a 95% confidence level when necessary. We continue evaluating the rankings generated by the algorithms using the mean reciprocal rank – MRR (as defined by Voorhees (1999)), since this metric awards words ranked near the top of the suggestion list which is appropriate for problems where there is no much space to show suggestions. Regarding space efficiency, the methods were evaluated according to the size of their indices in bytes, after discounting Java metadata overhead. We tested the methods with samples of 70k, 140k and 280k words extracted of our main dictionary. We took measurements as in the work by Boytsov (2011) for time efficiency, i.e, each experiment was performed twice and only the smaller time was recorded to minimize interference of OS daemons. All the experiments were performed on a 3.6 Ghz 6-core Intel i7 machine with 32 GB of RAM, Linux OS kernel 4.4.x, and JVM HotSpot 64-Bit Server.

4.3.3 Results

We start by defining an appropriate timeout parameter τ to prune processing, and best d value for AMPH-MML and AMPH-LHD. In all experiments reported in this chapter, if not mentioned differently, we used $k = 2$. These values were the best in our preliminary experiments, confirming results from Chapter 3. Figure 10 shows MRR and average response time for different timeout values for our methods using $d = 10$, the 140k dictionary, and EYE-PAIRS. AMMLSEP provides a baseline for MRR in this figure, shown as a red dashed horizontal line (MRR of about 72.11%).

As seen, AMPH-MML and AMPH-LHD has about the same performance of AMMLSEP regarding MRR, specially for $\tau \geq 64$ ms. AMPH-MML is slightly better than AMPH-LHD

Figure 10 – Timeout τ versus MRR and average response time for AMPH-LHD.

Source: (CRISTO *et al.*, 2017)

regarding MRR (about 1% gain, p -value < 0.05 , according to t-test) while AMPH-LHD is consistently faster than AMPH-MML. For $\tau = 64$ ms (magenta dashed vertical line), AMPH-LHD is 50% faster than AMPH-MML w.r.t. average response time (p -value < 0.01 , according to t-test). We observe the same relative performance for 70k and 280k dictionaries, using both EYE-PAIRS and ASWIKI-PAIRS datasets. From now on, all experiments reported with AMPH-MML and AMPH-LHD use $\tau = 64$ ms.

As for d , Table 8 shows the impact of deletions on MRR and average time for AMPH-MML, using both datasets with 70k dictionary. As seen, MRR reaches a peak value with about 6 deletions in ASWIKI-PAIRS and about 9 to 10 deletions in EYE-PAIRS. In ASWIKI-PAIRS, maximum MRR is achieved well before the maximum average time, which is obtained with 14 deletions. Similar relative performance is observed for AMPH-LHD and other dictionaries. Thus, we start to use $d = 6$ for ASWIKI-PAIRS and $d = 10$ for EYE-PAIRS in all experiments with AMPH-MML and AMPH-LHD, as this value is also used in our first experiments with AMMLSEP.

We evaluate the impact of the search pattern input size on average response time in the EYE-PAIRS dataset. This is a particularly important issue in real time input systems because the user expects to wait only an acceptable amount of time even after the input of a particularly large string. Figure 11 shows the input size versus the average response time obtained by AMPH-MML, AMPH-LHD, and AMMLSEP, using $\tau = 64$ ms and the 140k dictionary. We note that the Trie naturally avoided useless processing of spurious residuals of very large words. For instance, for words with more than 12 characters, usually resulting from errors on typing, processing time decreases as most of the residuals are not traversed in the Trie. In AMPH-MML and AMPH-LHD, such increase of time has to be controlled by pruning the processing of the residuals. As expected, the pruned versions never exceeds 64 ms.

To minimize the effect of the pruning strategy on the MRR performance of AMPH-LHD, we process string residuals sorted according to their number of deletions instead of a natural random order. Figure 12 shows the effect of processing strings ordered and not, according to the

d	EYE-PAIRS		ASWIKI-PAIRS	
	time (ms)	MRR	time (ms)	MRR
1	0.139 ± 0.01	0.376 ± 0.02	0.082 ± 0.01	0.395 ± 0.02
2	0.221 ± 0.01	0.521 ± 0.02	0.203 ± 0.02	0.618 ± 0.02
3	0.508 ± 0.05	0.616 ± 0.02	0.517 ± 0.01	0.702 ± 0.01
4	1.106 ± 0.13	0.670 ± 0.02	1.196 ± 0.03	0.724 ± 0.01
5	1.990 ± 0.20	0.698 ± 0.01	2.166 ± 0.05	0.729 ± 0.01
6	3.032 ± 0.28	0.713 ± 0.01	3.454 ± 0.10	0.730 ± 0.01
7	3.785 ± 0.33	0.724 ± 0.01	4.978 ± 0.16	0.730 ± 0.01
8	4.577 ± 0.38	0.727 ± 0.01	6.164 ± 0.22	0.730 ± 0.01
9	4.964 ± 0.41	0.729 ± 0.01	6.820 ± 0.28	0.730 ± 0.01
10	5.376 ± 0.44	0.729 ± 0.01	7.170 ± 0.32	0.730 ± 0.01
11	5.532 ± 0.45	0.728 ± 0.01	7.316 ± 0.34	0.730 ± 0.01
12	5.583 ± 0.46	0.729 ± 0.01	7.468 ± 0.35	0.730 ± 0.01
13	5.565 ± 0.46	0.728 ± 0.01	7.280 ± 0.34	0.730 ± 0.01
14	5.576 ± 0.46	0.728 ± 0.01	9.060 ± 9.06	0.730 ± 0.01

Table 8 – Deletions versus MRR and average response time for AMPH-MML (confidence intervals given for 95% confidence level).

Source: (CRISTO *et al.*, 2017)

number of deletions supported by AMPH-LHD, using the 70k dictionary and $\tau = 64$ ms.

Figure 11 – Input size versus response processing time.

Source: (CRISTO *et al.*, 2017)

Figure 12 – Deletions versus MRR for AMPH-LHD with processing of ordered and not ordered (random) string residuals in 70k dictionary.

Source: (CRISTO *et al.*, 2017)

As observed, random order leads to worse performance specially for a high value of d . With more deletions, more residuals can be obtained, increasing the number of candidates to evaluate. Thus, the possibility that the correct word is in the group that is evaluated before the timeout by chance is less likely. As a result, the performance of the random approach is increasingly worse. This effect is more prominent on EYE-PAIRS than on ASWIKI-PAIRS, since the pairs in ASWIKI-PAIRS has less errors per word to fix, yielding less candidates in general.

We compare our methods with baselines SISAP12 ($k = 1$ to 4) and AMMLSEP ($k = 1$ to 3, $d = 10$), considering the three dictionary sizes and the EYE-PAIRS and ASWIKI-PAIRS datasets. In particular, Figures 13 and 14 show the index size versus MRR. In these figures, for each method there are three MRR values corresponding to the use of the dictionaries with 70k, 140k, and 280k words, respectively.⁸

Figure 13 – Index size versus MRR (EYE-PAIRS).

Source: (CRISTO *et al.*, 2017)

Figure 14 – Index size versus MRR (ASWIKI-PAIRS).

Source: (CRISTO *et al.*, 2017)

We first observe that AMMLSEP index is larger than the ones from other methods, due to the use of a Trie instead of perfect hashing. In particular, regarding EYE-PAIRS, we also note that the error increases with the size of the dictionary. This happens because large dictionaries lead to more candidate words and, by extent, more false positives. It is also clear that the MML-based metric used by AMMLSEP, AMPH-MML and AMPH-LHD is worthwhile, resulting in consistent gains over SISAP12. AMMLSEP and AMPH-MML present equivalent levels of MRR while AMPH-LHD is just slightly inferior. When considering ASWIKI-PAIRS, we observe almost the same. However, in a scenario of low noise, SISAP12 is more competitive. We also note that dictionary size has a more pronounced impact on MRR, with larger dictionaries leading to more false positives. With fewer edit operations to consider (since ASWIKI-PAIRS had fewer errors per word, on average), the ability of MML-based metrics to recognize false positives is lower.

Regarding time efficiency, Figures 15 and 16 show the index size versus average response time. As before, for each method there are three values, corresponding to 70k, 140k, and 280k dictionaries respectively. In both figures, AMPH-MML and AMPH-LHD present performance between those of methods SISAP12 and AMMLSEP. For AMPH-MML and AMPH-LHD, the performance is also less dependent on dictionary size. That happens because AMPH-MML and AMPH-LHD do not need to observe all possible common residuals between w and v to estimate the distance between two strings w and v , but only the first one, unlike SISAP12 and AMMLSEP.

⁸ We were not able to run SISAP12 with the 280k dictionary, $k = 4$, using the same VM heap size limit adopted for the other methods.

AMPH-LHD is consistently twice as faster than AMPH-MML. Compared to AMMLSEP, it is about 50 to 170 times faster, depending on dictionary size and k . It is even faster than SISAP12, $k = 4$, in EYE-PAIRS dataset. Considering the high MRR obtained by AMPH-MML and AMPH-LHD, they are more effective in word recognition than SISAP12 and faster than AMMLSEP with little to no loss in recognition quality.

Figure 15 – Index size versus time (EYE-PAIRS).

Source: (CRISTO *et al.*, 2017)

Figure 16 – Index size versus time (ASWIKI-PAIRS).

Source: (CRISTO *et al.*, 2017)

4.4 Final considerations

In this chapter we presented two methods to recognize words in very noisy application scenarios. AMPH-MML retrieves candidates by combining an asymmetrical Mor-Fraenkel index with a minimum perfect hashing. It then ranks the candidates using a MML-based distance metric. AMPH-LHD extends AMPH-MML substituting the MML metric with a linear heuristic which benefits from the processing order of the residuals and information stored in the Mor-Fraenkel index. In particular, the ordered processing allows that the more promising candidates are evaluated earlier, enabling prune strategies based on limiting processing time. By testing our methods using word recognition and spelling correction datasets, we have shown that the proposed methods are very effective. They largely outperform AMMLSEP regarding time efficiency, being 25 to 170 times faster, depending on k and dictionary size. Regarding word recognition precision on noisy scenarios, AMPH-MML achieved the same precision of AMMLSEP while AMPH-LHD was just slightly inferior. These results were reported in the paper “Fast Word Recognition for Noise channel-based Models in Scenarios with Noise Specific Domain Knowledge.” (CRISTO *et al.*, 2017), published in the Proceedings of the 2017 ACM Conference on Information and Knowledge Management – CIKM 2017.

IMPROVEMENTS ON WORD RECOGNITION

In spite of its effectiveness, AMPH-LHD is based on a simple balanced combination of a language model and a noise model. Moreover, its noise model does not include an important noise source, the eye movements variability. As observed by [Beers \(2007\)](#) and [Beers \(2008\)](#), such variability is strongly correlated with movement speed, distance and obliqueness, parameters associated with the context of the character being typed. In this chapter we improve AMPH-LHD by taking into consideration contextual information associated with the eye movements. We argue that this information may capture errors associated with eye searching behavior and eye movement variability.

We also study how to control the balance between the language and the noise models. Given that these models are not directly comparable, it is important to investigate the influence of each of them on the performance of the combined model. We then study the impact of a smoothing technique and a mode filter on the input stream. Many action misunderstands derived from fast eye movements, eye tracker imprecision and jitter can be avoided by using simple smoothing techniques. Although we have used such techniques along the noise models proposed in the past chapters, no study was provided about their combination.

5.1 Revisiting our Word Recognition model

As our previous methods, we consider that the typing of s when the user intended to type w is equivalent to the transmission of w through a noisy channel as studied by [Kernighan, Church & Gale \(1990\)](#) and [Mays, Damerau & Mercer \(1991\)](#) in the context of spelling correction. Thus, to recognize a word w from the observed scanpath s , the receiver has to determine, among possible candidate words $t \in \mathcal{D}$, which one is more likely the original signal w by observing s . This recognition process can be translated, using Bayesian inference, into Eqs. [5.1](#) and [5.2](#) (log space):

$$\hat{w} \approx \arg \max_{t \in \mathcal{D}} P(s|t)P(t)^\lambda \quad (5.1)$$

$$\hat{w} \approx \arg \max_{t \in \mathcal{D}} \log P(s|t) + \lambda \log P(t) \quad (5.2)$$

where s is the input pattern, $t \in \mathcal{D}$ is a dictionary word, η is a normalization constant, $P(s|t)$ indicates how likely s could be a distorted version of t , $P(t)$ is the prior probability of t , and λ is a balance power factor. This equation has two main components, the language model $P(t)$ and the noise model $P(s|t)$. $P(t)$ can be estimated from a language corpus to define the t prior probability. $P(s|t)$ can be estimated by any similarity metric between the received signal s and a candidate word t . Since $P(t)$ and $P(s|t)$ are often not directly comparable, λ can be used to leverage them, as in [Jurafsky & Martin \(2009\)](#). By defining $\lambda = 1$, we obtain the same model proposed in Chapter 3 (AMMLSEP).

5.1.1 Noise model

In our previous models, we adopted a less error-centric view by considering the importance of the characters that were not modified. Thus, by including matching operations, $P(s|t)$ can be estimated as the most probable sequence of operations to transform s into t , independently if they are associated with errors. In addition, $P(s|t)$ can be viewed as correlated with the costs associated with character matching, substitution, inclusion, and deletion, as well as the transposition of adjacent characters (cf. Eqs. 3.1 to 3.5).¹ According this approach, the edit probabilities are context independent and, as we will see in next sections, this is not the case for some kinds of noise. Thus, we can redefine edit probabilities as follows:

$$\delta_{s[i] \rightarrow \epsilon} = P_d P_E(t_{[j]} s[i] \rightarrow t_{[j]} \mid s_{[i-1]}) \quad (5.3)$$

$$\delta_{\epsilon \rightarrow t_{[j]}} = P_i P_E(t_{[j-1]} \rightarrow t_{[j-1]} t_{[j]} \mid s_{[i-1]}) \quad (5.4)$$

$$\delta_{s[i] \rightarrow t_{[j]}} = P_s P_E(s[i] \rightarrow t_{[j]} \mid s_{[i-1]}) \quad (5.5)$$

$$\delta_{s_{[i-1:i]} \rightarrow s_{[i:i-1]}} = P_t P_E(s_{[i-1:i]} \rightarrow s_{[i:i-1]} \mid s_{[i-2]}) \quad (5.6)$$

In Eqs. 5.3 to 5.6, the conditional notation $P_E(e|c)$ is adopted to indicate the probability of each specific edit operation e , given context c . Thus, for instance, $P_E(s \rightarrow c|a) > P_E(s \rightarrow c|p)$ if the substitution of s by c is more probable after letter a than after letter p . We assume for the cases where context cannot exist ($s_{[i-1]}$, $i \leq 1$ and $s_{[i-2]}$, $i \leq 2$) that the (previous) context is given by the start of the string indicated by symbol “ $\angle$ ”.

The probability of a specific edit operation e ($P_E(e|c)$ if context is taken into account) should consider all the sources of errors associated with eye-typing. Since an eye typing system

¹ As in Chapter 3, for clarity of exposition, we omit logarithms in the equations.

probably shares much of the error sources found in any other typing system, we can estimate this probability considering that the edit operation is derived from a common source of errors or from an eye-typing specific source. Thus, $P_E(e|c)$ can be given by Eq. 5.7:

$$P_E(e|c) = \alpha P_C(e|c) + (1 - \alpha) P_N(e|c) \quad (5.7)$$

where $P_C(e|c)$ and $P_N(e|c)$ are the probabilities associated with common and specific eye-typing noise, respectively, and α is a factor used to balance the importance of each source of evidence.

$P_C(e|c)$ can be extracted from a large enough corpus of common typing errors. As we presented in Section 2.1, these errors are related to the following sources: (i) *physical factors* such as the distance between the keys; (ii) *visual factors* such as visual similarity of characters, position in a word, and occurrence repetition; (iii) *phonological factors* such as phonological similarity of characters/words; and (iv) *cognitive factors* such as language domain and vocabulary familiarity. As the corpus of errors has no estimates for matches, by setting $\alpha = 1$ and assuming that edit probabilities are context independent, $P_E(e|c) \approx P_E(e)$ corresponds to the same estimates used by Norvig (2009).

Unlike $P_C(e|c)$, a large enough corpus is hardly available to estimate $P_N(e|c)$. Thus, $P_N(e|c)$ can be estimated from particular characteristics of the noise observed in eye typing and presented in Chapter 2. For $\alpha < 1$, such estimates become relevant and are discussed in the following sections.

5.1.2 Sources of error in eye-based typing

Among the noise sources associated with eye typing and discussed in Chapter 2, we revisit the eye tracker device and eye movement variability.

5.1.2.1 Eye-tracker measurement

Different from what occurs in very accurate trackers and well controlled conditions, in inexpensive eye-trackers the accuracy and precision of the device are the mainly sources of noise. Considering only the tracker noise, the probability that a wrong character c substitutes target b or is inserted before/after b is inversely proportional to the distance between c and b . Supposing an interface defined by a discrete set of points (x, y) , $x \in X$ and $y \in Y$, the probability of an arbitrary point $\mathbf{p} = (x_i, y_i)$ be taken as a point of b can be estimated by the bi-variate distribution given by Eq. 5.8:

$$P_T(\mathbf{p}|b) = \mathcal{N}_{\mathbf{p}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_0) \quad (5.8)$$

Figure 17 – Error ellipsis (shaded area) resulting from sensory noise observed in saccade. Saccade is represented by a vector from point a (the center of the source key, represented by a circle with diameter D) to point b (the center of the D -diameter circle representing the target key).

where mean vector $\mu = (0,0)$ corresponds to the center of b and co-variate matrix $\Sigma_0 = \text{diag}(\sigma^2, \sigma^2)$.

5.1.2.2 Eye movements variability

Saccades and fixations are the eye movements most relevant for typing. Usually fixations are modelled as a simple Gaussian component. However, saccadic eye movements are more complex to model. According to Beers, sensory noise is the main cause of the variability. That noise results from uncertainty in the localization of targets which are out of the fovea region, due to low density of cells in retina. This results in a variability in the saccade endpoints which can be decomposed into two components: the variability in movement amplitude (σ_R) and in movement direction (σ_ϕ). Psychophysical studies have shown that (1) the localization of peripheral targets is more precise in the ϕ direction than in the R direction which results in an error ellipsis; (2) it is much better along the horizontal and vertical meridians than in oblique directions, i.e., it is dependent on direction θ of the saccade vector (the vector from the start to the endpoint of the intended saccade); and (3) it varies across the visual field in the same way as visual acuity does, i.e., it also depends on movement amplitude E . These factors, illustrated in Figure 17, are modelled by Beers as follows.

Visual acuity decreases with movement amplitude E . Considering component R , it can be approximated as $CV_R^h(E) = CV_R'(1.95e^{-E/2.65} + 1)$, where E is given in degrees and CV_R' is the CV at large amplitudes. Along the horizontal axis, CV_R' is estimated as 0.045. $CV_\phi^h(E)$ can be derived from $CV_R^h(E)$ as it is known that $CV_R^h/CV_\phi^h \approx 6$ for targets on the horizontal and vertical meridians, and $CV_R^h/CV_\phi^h \approx 2$ for targets in other directions.

The effect of θ on precision is estimated, in components $\omega = \phi$ and $\omega = R$, as $CV(\theta) = CV_\omega^h(1 + V \sin^2 \theta)$, where CV_ω^h represents the precision on the horizontal meridian, and θ represents the saccade direction. Factor V is 0.30 for targets above the horizontal axis and 0.56 for targets below that axis. The diameter D of the target disk also contributes for sensory noise.

Assuming an equal probability for any position on this disk, an additional variance of $\frac{1}{12}D^2$ in both the horizontal and vertical directions is included. Finally, we define σ_R and σ_ϕ as:

$$\sigma_R = E\left(\frac{1}{12}D^2 + CV_R^h(1 + V \sin^2 \theta)\right) \quad (5.9)$$

$$\sigma_\phi = E\left(\frac{1}{12}D^2 + CV_\phi^h(1 + V \sin^2 \theta)\right) \quad (5.10)$$

Thus, by taking into account only the sensory noise, the probability of an arbitrary point $\mathbf{p}$ be mistaken as a point of key b can be estimated as the bi-variate distribution given by Eq. 5.11.

$$P(\mathbf{p}|a, b) = \mathcal{N}_{\mathbf{p}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_1) \quad (5.11)$$

where a is the startpoint of the saccade (context key), mean vector $\boldsymbol{\mu} = (0, 0)$ corresponds to the center of b and $\boldsymbol{\Sigma}_1$ is given by Eq. 5.12:

$$\boldsymbol{\Sigma}_1 = \kappa \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} \sigma_R^2 & 0 \\ 0 & \sigma_\phi^2 \end{pmatrix} \quad (5.12)$$

In Eq. 5.12, the Gaussian distribution is rotated according to θ and κ is a factor which increases the error associated with the sensory noise. Factor κ is necessary because the model previously described is based only on saccades which start with eye muscles in a rest position (primary saccades). As consequence, by adopting $\kappa = 1$, we get an optimistic model since non-primary saccades are more likely in eye typing. On the other hand, by using $\kappa > 1$, we simulate a more likely outcome, that is, larger errors.

5.1.3 Probabilities of Specific Edit Operations according to Noise Sources

All previous sources of errors will increase the possibility that the eye misses the target key b and points out to a position $\mathbf{c}$ in its neighborhood. When considering only sensory noise, such error also depends on the point $\mathbf{a}$ where the eye movement starts from. The combined effect of these sources can be estimated as a Gaussian mixture model. Thus, given points $\mathbf{a}$ and $\mathbf{b}$, the center of source key a and target key b respectively, the probability of mistyping c in place of b is:

$$P(c|\mathbf{a}, \mathbf{b}) = \sum_{\forall \mathbf{c} \in c} \pi_0 \mathcal{N}_{\mathbf{c}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_0) + \pi_1 \mathcal{N}_{\mathbf{c}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_1) \quad (5.13)$$

where π_0 and π_1 , $\pi_0 + \pi_1 = 1$, indicate the relative importance of each individual source, and $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}_0$ and $\boldsymbol{\Sigma}_1$ are the same as in Eqs. 5.8 and 5.11.

Figure 18 – Saccade endpoint probabilities when (a) target key is c, disregarding saccade startpoint ($\pi_0 = 1, \pi_1 = 0$); (b) target key is c in saccade originated in p ($\pi_0 = 2, \pi_1 = 1, \kappa = 3$); and (c) target key is d in saccade originated in f ($\pi_0 = 2, \pi_1 = 1, \kappa = 3$). All probabilities were sampled assuming a tracker error of 1 degree at 95% confidence level ($2\sigma_0 = 1^\circ$).

Other components can be included in the Gaussian mixture to represent additional sources of error, such as the search behavior discussed in Section 2.1. These sources can be also approximated by tuning parameters π_0 , π_1 , and κ , as in the bi-variate distributions derived from Eq. 5.13 and illustrated in Figure 18. In particular, Figure 18(a) shows the error distribution around target key c when only tracker noise is considered ($\pi_0 = 1$). Figure 18(b) shows the error distribution around c, after an oblique eye movement originated at p, when tracker and sensory noise are taken into account, $\pi_0 = 2$, $\pi_1 = 1$, and $\kappa = 3$. As result, error probabilities increase along the axis from p to c. Finally, Figure 18(c) shows the error distribution around d when context is a close key, f, and other parameters are the same. We note that for a very short and horizontal saccade, error probabilities are closely concentrated around the target.

We can now estimate specific edit probabilities by sampling points using Eq. 5.8 and the layout of the visual keyboard, resulting in Eqs. 5.14 to 5.16, when context is *not* considered. Note that Eqs. 5.14 and 5.15 are the same presented in Section 3.1.

$$P_N(t_{[j]}s_{[i]} \rightarrow t_{[j]}) = \eta_0 \sum_{\forall \mathbf{p} \in s_{[i]}} P_T(\mathbf{p}|t_{[j]}) \quad (5.14)$$

$$P_N(s_{[i]} \rightarrow t_{[j]}) = \eta_1 \sum_{\forall \mathbf{p} \in s_{[i]}} P_T(\mathbf{p}|t_{[j]}) \quad (5.15)$$

$$P_N(s_{[i-1:i]} \rightarrow s_{[i:i-1]}) = \eta_2 \sum_{\forall \mathbf{p} \in s_{[i]}} P_T(\mathbf{p}|s_{[i-1]}) \quad (5.16)$$

The contextual versions of these equations are derived from sampling using Eq. 5.13, resulting in:

$$P_N(t_{[j]}s_{[i]} \rightarrow t_{[j]} \mid s_{[i-1]}) = \eta_0 P(s_{[i]} \mid \mathbf{s}_{[i-1]}, \mathbf{t}_{[j]}) \quad (5.17)$$

$$P_N(s_{[i]} \rightarrow t_{[j]} \mid s_{[i-1]}) = \eta_1 P(s_{[i]} \mid \mathbf{s}_{[i-1]}, \mathbf{t}_{[j]}) \quad (5.18)$$

$$P_N(s_{[i-1:i]} \rightarrow s_{[i:i-1]} \mid s_{[i-2]}) = \eta_2 P(s_{[i]} \mid \mathbf{s}_{[i-2]}, \mathbf{s}_{[i-1]}) \quad (5.19)$$

where η_0 to η_2 are normalization constants taken over deletions, substitutions, and transpositions, respectively. Further, we can treat deletions at the start of the words as special cases. Without additional information, the probability of a key being mistyped is proportional to how many keys are in its neighborhood. For instance, in a QWERTY layout, a character located at the center (e.g., g) is more likely to be replaced by another character than one located at the corner (e.g., p). Based on this observation, we can estimate deletions at the start of the word as:

$$P_N(\langle s_{[i]} \rightarrow \langle) = \eta_3 \sum_{\forall b \in \Sigma} \sum_{\forall \mathbf{p} \in s_{[i]}} P_T(\mathbf{p} \mid b) \quad (5.20)$$

$$P_N(\langle s_{[i]} \rightarrow \langle \mid \langle) = \eta_3 \sum_{\forall b \in \Sigma} \sum_{\forall \mathbf{p} \in s_{[i]}} P_T(\mathbf{p} \mid b) \quad (5.21)$$

where η_3 is a normalization constant taken over all deletions at the start of the words.

Regarding matches, given two strings s and t , the probability of finding a spurious match between two arbitrary characters of s and t is larger for more frequent characters. Thus, we estimate $P_N(s_{[i]} \rightarrow s_{[i]})$ as the inverse of the popularity of $s_{[i]}$ in the language. Finally, we consider that the remaining probabilities are the same as observed in a general corpus.

5.1.4 Noise model Estimation

In Chapter 4, to calculate $P(s \mid t)$ to all words of a possibly large dictionary in a feasible time, our model – AMPH-LHD – (i) smoothed the eye-tracker input signal to minimize jitter and key repetitions; and (ii) combined a fast index search with a reduced search space. In particular, the first technique was done by processing all the points captured by the eye tracker with the saccade detection and fixation smoothing algorithm proposed by [Kumar, Winograd & Paepcke \(2007\)](#). In addition, to avoid an excess of key repetitions in s , specially due to keys seen during saccades, a mode filter was applied, as explained follows. Each key captured by the eye tracker is counted up to $2M - 1$ observations are made. At this point, the most frequent key is concatenated to s if it has been observed more than $M - 1$ times and the first $M - 1$ observations are forgotten. Let $\bar{x}$ be the frequency moving average of the last 3 observed keys. Any key observed more than $\psi \bar{x}$ times is considered to be duplicate and concatenated twice to s . The second technique was adapted to support contextual specific edit probabilities as we describe below.

5.1.4.1 Estimation Algorithm

The main strategy to get fast estimates consists in reducing the search space to include only strings $t \in \mathcal{D}$, which are promising candidates, i.e., strings t such that $\mathcal{E}_{DEL}(s, t) \leq d$ and $\mathcal{E}_{IST}(s, t) \leq k$. As we saw in past chapters, this can be accomplished by using an asymmetrical Mor-Fraenkel index. AMPH-LHD benefits from the use of a perfect hashing function which allows search the input string residuals in $O(1)$ time. Moreover, since the algorithm supports pruning, the complete recognition operation (from neighborhood acquisition to probability estimation, including the index retrieval) can be limited to a maximum time τ . Finally the method uses a linear heuristic that is used to estimate $P(s|t)$ that take advantage of index stores information (all characters in t that also occur in s). In general, assuming that $(s_{[i]}, t_{[j]})$ is the current pair of characters to be aligned and transpositions are not allowed, one of the following situations is possible:

- $i \notin P^s \wedge j \notin P^t$: edit operation is a match of $s_{[i]}$ and next pair to inspect is $(s_{[i+1]}, t_{[j+1]})$.
- $i \in P^s \wedge j \in P^t$: edit operation is a substitution of $s_{[i]}$ by $t_{[j]}$ if $s_{[i]} \neq t_{[j]}$, otherwise is a match of $s_{[i]}$. The next pair to inspect is $(s_{[i+1]}, t_{[j+1]})$.
- $i \notin P^s \wedge j \in P^t$: edit operation is insertion of $t_{[j]}$ and next pair to inspect is $(s_{[i]}, t_{[j+1]})$.
- $i \in P^s \wedge j \notin P^t$: edit operation is deletion of $s_{[i]}$ and next pair to inspect is $(s_{[i+1]}, t_{[j]})$.

Transpositions can be supported by checking if the current operation is not part of a sequence of operations equivalent to a transposition. In that case, the sequence is just replaced by the transposition. The complete algorithm, supporting transpositions and edit operations with previous context, is shown in Algorithm 5. From now on we refer to it as LHDctx.

Unlike LHDistance, defined in Chapter 4, LHDctx keeps track of the context information related to the eyes' movements. In fact, it uses two distinct context information variables, ρ and c . Variable ρ saves the previous character, believed to be the correct one. It provides context for a possible following deletion or insertion operation. Variable c records the origin of the eye movement. The initial context is always the start of the word ($\langle$). Note we do not consider the origin of the eye movement before the start of the word because we want to get estimates agnostic regarding what happens in the interface beyond word typing. As result, c records eye origin positions only after ρ indicates that the previous context is not the start of the word anymore (line 7).

To illustrate parameters ρ and c , suppose a user typing vip misinserts o, yielding $s = viop$. To correct the error, o has to be deleted. After such deletion, and following the work by Norvig (2009), the context for the insertion of p is $\rho = i$, resulting in $i \rightarrow ip$. However, the visual context for this insertion is $c = o$, since this was the last character really seen by the user. The resulting probability would be $P(i \rightarrow ip|o)$, that is, the probability of inserting p after i, considering that the eye moved from o. Parameters ρ and c depend on the inferred edit

Algorithm 5: Linear heuristic

```

1 Function LHDctx( $s, C^t, P^s, P^t$ )
2    $\mathcal{E} \leftarrow []$ ;                                /* List of edit operations */
3    $i \leftarrow j \leftarrow 0$ ;                      /* Current char pos at  $s$  and  $t$  */
4    $m \leftarrow n \leftarrow 0$ ;                      /* Current pos at  $P^s$  and  $C^t/P^t$  */
5    $\rho \leftarrow \langle$ ;                             /* Previous key assumed correct */
6    $c \leftarrow \langle$ ;                             /* Previous seen key - context */
7    $\beta_{xy} \leftarrow \text{return } \langle \text{if } x = \langle \text{ else } y$ ;
8    $K(e) \leftarrow \text{return kind of edit oper. } e (i, d, s, t, \text{ or } m)$ ;
9   while  $i < |s|$  do
10    if  $i = P_m^s \wedge j = P_n^t$  then
11      if  $s_{[i]} = C_n^t$  then
12         $\rho, c \leftarrow \text{AddMatch}(\mathcal{E}, s_{[i]}, c)$ ;          /* Match */
13      else
14         $\rho, c \leftarrow \text{AddSubsOrTrans}(\mathcal{E}, s_{[i]}, C_n^t, c)$ ; /* Substitution */
15       $i++, j++, m++, n++$ ;
16    else if  $i = P_m^s$  then
17       $\rho \leftarrow \text{AddDelOrTrans}(\mathcal{E}, \rho, s_{[i]}, \beta_{\rho c})$ ; /* Delete  $s_{[i]}$  */
18       $i++, m++$ ;
19    else if  $j = P_n^t$  then
20       $\rho, c \leftarrow \text{AddInsOrTrans}(\mathcal{E}, \rho, C_n^t, \beta_{\rho c})$ ; /* Insert  $C_n^t (= t_{[j]})$  */
21       $j++, n++$ ;
22    else
23       $\rho, c \leftarrow \text{AddMatch}(\mathcal{E}, s_{[i]}, c)$ ;
24       $i++, j++$ ;
25  while  $n < |C_n^t|$  do
26     $\rho, c \leftarrow \text{AddInsOrTrans}(\mathcal{E}, \rho, C_n^t, \beta_{\rho c})$ ; /* Insert remaining  $t$  chars */
27     $n++$ ;
28  return  $\prod_i P_{K(\mathcal{E}_i)} P_E(\mathcal{E}_i)$ ;

```

operation. Thus they are defined in functions `AddMatch`, `AddSubsOrTrans`, `AddDelOrTrans`, and `AddInsOrTrans` (cf. Algorithm 6).

All pairs $(s_{[i]}, t_{[j]})$ are aligned from line 9 to 27. Substitutions, deletions and insertions are added to $\mathcal{E}$ only after they had been verified not to be part of a transposition (functions `AddSubsOrTrans`, `AddDelOrTrans`, and `AddInsOrTrans`). After determining all the edit operations, the edit script probability is calculated according to Eqs. 5.3 to 5.6 (line 28). Note that, given strings s and t , *LHDCtx* estimates $P(s|t)$ for a single pair of residuals (s', t') . As the linear heuristic from AMPH-LHD, *LHDCtx* is estimated for up to L pairs, ordered according to how promising they are (using as heuristic that pair (s', p') is as promising as $|s| - |s'|$ is close to zero).

From now on, we refer to our algorithm, based on *LHDCtx*, as C-AMPH-LHD.

Algorithm 6: Functions AddMatch, AddDelOrTrans, AddSubsOrTrans, and AddInsOrTrans.

```

1 Function AddMatch( $\mathcal{E}, c_1, ctx$ )
2   |   add ( $c_1 \rightarrow c_1|ctx$ ) to  $\mathcal{E}$ ;
3   |   return  $c_1, c_1$ ;
4 Function AddSubsOrTrans( $\mathcal{E}, c_1, c_2, ctx$ )
5   |   if  $\mathcal{E}_{last} = c_2 \rightarrow c_1$  then
6   |   |    $\mathcal{E}_{last} \leftarrow c_2 c_1 \rightarrow c_1 c_2|ctx$ ;
7   |   else
8   |   |   add ( $c_1 \rightarrow c_2|ctx$ ) to  $\mathcal{E}$ ;
9   |   return  $c_2, c_1$ 
10 Function AddDelOrTrans( $\mathcal{E}, c_1, c_2, ctx$ )
11 |   if  $\mathcal{E}_{last-1} = k \rightarrow kc_2 \wedge k \neq \prec \wedge c_1 \neq c_2$  then
12 |   |    $\mathcal{E}_{last-1} \leftarrow c_1 c_2 \rightarrow c_2 c_1|ctx$ ;
13 |   |   delete  $\mathcal{E}_{last}$ ;
14 |   else
15 |   |   add ( $c_1 c_2 \rightarrow c_1|ctx$ ) to  $\mathcal{E}$ ;
16 |   return  $c_1, c_2$ 
17 Function AddInsOrTrans( $\mathcal{E}, c_1, c_2, ctx$ )
18 |   if  $\mathcal{E}_{last-1} = kc_2 \rightarrow k \wedge k \neq \prec \wedge c_1 \neq c_2$  then
19 |   |    $\mathcal{E}_{last-1} \leftarrow c_2 c_1 \rightarrow c_1 c_2|ctx$ ;
20 |   |   delete  $\mathcal{E}_{last}$ ;
21 |   else
22 |   |   add ( $c_1 \rightarrow c_1 c_2|ctx$ ) to  $\mathcal{E}$ ;
23 |   return  $c_2, c_1$ 

```

5.2 Experiments

5.2.1 Baselines

Our baseline in this chapter is AMPH-LHD with the best parameters we found so far: $L = 1$, $\tau = 64$ ms, $M = 5$, and $\psi = 2$ for a 30Hz eye tracker.

5.2.2 Evaluation Methodology

As our list of noisy words for recognition, we used the dataset presented in Section 3.3 as well as the estimated probabilities for P_i , P_d , P_s , P_t , and P_m . The dataset contains 3,073 pairs of strings s and t , where s is the mode filter output and t is the word the user intended to type. The probability of specific edit operations was obtained in the same way as in previous chapters — noise specific information from eye-typing logs collected by us and frequencies of edit operations observed in the general corpus used by Norvig (2009). Again, the dictionaries with word frequencies and frequencies from edit operations were the ones provided by Norvig (2009).

We used reciprocal rank metric as defined by Voorhees (1999) to evaluate the methods according to their ranking quality. All experiments were performed on a 3.47Ghz 6-core i7 machine with 16GB of RAM. We used Linux OS, kernel 3.13.x, and Java 1.8, JVM HotSpot 64-Bit Server. To ensure that the differences among the methods we compare are statistically significant, along with the means of our evaluation metrics, we report its confidence intervals. We adopt a confidence level of 95%, except in cases in which the reader is explicitly notified. We also report p-values obtained with paired t-test as additional evidence in some cases (GREENLAND *et al.*, 2016).

5.2.3 Results

We begin by defining τ and L parameters. The higher their values, the more accurate is the resulting ranking. Since in Chapter 4, we were more concerned with the efficiency of response time, we set them to low values ($\tau = 64$ ms and $L = 1$). However, we observed a very comfortable user experience even with higher values in our tests with interface that resulted in gains in terms of MRR. This can be seen in Figure 19 which shows the MRR values obtained with AMPH-LHD and C-AMPH-LHD using $L = 5$ and τ ranging from 64 to 400 ms.

Figure 19 – Maximum pruning time (τ) x MRR for AMPH-LHD and C-AMPH-LHD.

Figure 20 – Effect of outlier and duplicate letter detection parameters on MRR.

As observed, the MRR values are better for higher values of τ . In addition, even using a longer pruning time and analyzing up to 5 pairs of residual strings per Mor-Fraenkel index entry, the mean response time of both algorithms is only 10 ms, well below the mean eye fixation time of 200 ms (BEERS, 2007). We also observe in the figure that the eye variability context information was not very useful when considering this initial parameter setting.

Using τ and L , we now define the best values for parameters M and ψ . As previously described, these parameters define (i) the minimum number of times a letter has to be observed in a time window of size $2M - 1$ to not be considered an outlier, and (ii) for how much time a letter should be observed (in relation to the moving average of the last three observations) to be considered a repetition. Figure 20 shows the gain in MRR for different values of parameters M and ψ (the darker is the color, the larger is the MRR). We note in this figure that the best MRR

values are obtained for $M = 4$ and $\psi = 3.0$, independently whether contextual probabilities are used.

These results indicate that a letter is not considered as an outlier nor a letter “seen” during saccade if it is observed at least $M = 4$ times in 7 observations taken every 33 ms (for 30 Hz eye tracker). That letter will be considered as repeated if the user look at it $\psi = 3$ times longer than she did, in average, for the last three previously observed keys.

We now use parameters τ, L, M and ψ to investigate the best α value for the model based on contextual probabilities, as illustrated by Figure 21. As in Chapter 3, we observe that, for a non-contextual model, $\alpha = 0.7$ provides the best value of MRR. This indicates that to achieve the best results, the noise model needs to consider eye-typing specific errors as well as general domain errors. For the contextual model, however, the best MRR is achieved for $\alpha = 0.006$. This small value does not mean that general errors are useless when contextual probabilities are adopted. It results from the smaller estimates of contextual probabilities since there are many more events to observe. For instance, when considering contextual probabilities, the single event “look at X” corresponds to the events “look at the X, given the eye has moved from Y”, for every possible Y. As consequence, each event is much less likely and a smaller α value is necessary to balance the importance of specific eye-typing and general errors.

Figure 21 – Best alpha for methods using or not contextual probabilities.

Figure 22 – MRR x importance of the language model (λ).

We also note that, based on our initial setting, no important gain is observed by using different settings for τ, L, M and ψ . Again, the initial results indicate that with contextual probabilities, the model performs worse. This situation, however, is completely different when we modify the influence of the language model (parameter λ), as observed in Figure 22.

We now observe that C-AMPH-LHD model, based on contextual probabilities, outperforms the basic AMPH-LHD for $\lambda \geq 2$. C-AMPH-LHD achieves peak performance (MRR = 80.2%) at $\lambda = 2.4$. It outperforms AMPH-LHD even at its peak performance ($\lambda = 2$, MRR = 79.2%) with a gain of 1.3%. Although the confidence intervals overlap, a paired t-test using the same data results in a p -value = 0.0001. This suggests that confidence intervals will not overlap for a larger sample size. In fact, after comparing the same methods for the 11 thousand input

strings of our largest experiment (cf. Chapter 6) we obtained no-overlapping confidence intervals and a gain of 1.6% (C-AMPH-LHD MRR = 81.32 ± 0.0062 ; AMPH-LHD MRR = 79.97 ± 0.0064). Both models present better performance for $\lambda > 1$, which indicates that the original AMPH-LHD ignores much useful information provided by the language model. Considering the best λ , we can now compare again the variation of MRR according to α for the models using or not context probabilities, including the original AMPH-LHD. Such comparison is shown in Figure 23.

Figure 23 – Best alpha for proposed methods and CIKM’17 AMPH-LHD.

As we can see, the best MRR resulting of the combination of contextual probabilities and the language model is 0.802.² Compared with the original AMPH-LHD (MRR = 0.728), it represents a gain of 10.16%. With this ranking, we expect to increase the placement of correct words at top positions in the list of suggestions, minimizing the user’s need to switch between screens when looking for a fix. This is particularly important since context changes imply additional task for the eyes. Table 9 illustrates the ability of C-AMPH-LHD to rank correct words at top positions, by detailing ranking information for our test set of 3,073 pairs of strings. The table shows the proportion of pairs which were ranked at top 1, 6, 12, and 18 positions.

Table 9 – Proportion of the 3,073 pairs in which correct word was placed at top-1, 6, 12, and 18 positions.

method	Top-1	Top-6	Top-12	Top-18
C-AMPH-LHD	73.68 ± 1.56	89.72 ± 1.07	92.06 ± 0.96	93.00 ± 0.90
AMPH-LHD	64.85 ± 1.69	83.89 ± 1.28	87.24 ± 1.15	89.19 ± 1.07

As we can see, by using C-AMPH-LHD ($\lambda = 2.4$, $\alpha = 0.006$, $M = 4$, $\psi = 3$, $L = 5$ and $\tau = 400$ ms), we expect the user will find a correct suggestion often at the first page, switching screens less frequently than with AMPH-LHD. Its gain, in Top-1, over AMPH-LHD was 13%. In Top-6, the gain was smaller, achieving 7%.

² The best MRR, without context probabilities for $\lambda = 2.4$, is 0.783. Again, the difference is significant according to t-test and intervals do not overlap in larger sample size.

5.3 Final Considerations

In this chapter, we improved words recognition for an eye typing environment. To accomplish that, we (i) investigated how to balance the influence of the language and noise components of a noisy-channel based model; (ii) extended its noise component to include information associated with the eye movements to capture errors related to movement variability; and (iii) studied the impact of preprocessing the input stream to filter unintended selections made during fast eye movements and diminish the jitter characteristic of inexpensive eye tracker devices. In an off-line experiment performed with eye-typing interaction logs, our results showed that all these strategies had little impact when used in isolation. However, when combined, they enabled our model (C-AMPH-LHD) to significantly outperform our previous word recognizer (AMPH-LHD) with a gain of 10.16% in MRR. These results are part of the journal article “A Noisy-channel based Model to Recognize Words in Dwell-free Eye Typing Systems”, submitted to the ACM Transactions on Computer-Human Interaction, and currently under review.

EVALUATION ON A REAL INPUT SYSTEM

Despite the positive results achieved by our method to better rank words, we still need insights on how the user performance is affected in terms of typing speed and fatigue related to typing errors. Evaluations on log data are restricted since they do not capture the reactions of the users to the system. They are also optimistic because they assume that the user is always willing to lookup suggestions independently on their ranking positions, which is unlikely. Thus, although effective, C-AMPH-LHD has to be integrated into a complete eye typing system to evaluate its impact on a real-time transcription task. In this chapter we incorporate C-AMPH-LHD in the dwell-free interface Filtertypedping developed by [Pedrosa *et al.* \(2015\)](#) and provide a comprehensive evaluation on how different typing actions can be impacted by an improved word recognizer. As far as we know, there is no detailed study about how a change in the ranking of words can influence the user behaviour and affect writing, selection, and deletion actions independently.

In this chapter, we refer to the original Filtertypedping by its name while we call our modified version of Filtertypedping as C-AMPH-LHD. We expect that our new method will (1) improve user experience, since it would be easier for users to select the desired words; and (2) increase typing speed, since users would not waste time redoing actions and will be able to perform frequent actions faster.

6.1 Evaluation methodology

To evaluate C-AMPH-LHD, we adopted the same methodology used by [Pedrosa *et al.* \(2015\)](#). In particular, the resulting study involved 8 participants, 3 female and 5 male. The participants were 23.37 years old on average (std = 1.06, min = 22, max = 25). All the participants were: (a) familiar with the QWERTY keyboard, (b) without cognitive impairments or physical disabilities that could prevent them from using the tool (e.g. visually impaired individuals), and (c) fluent in the English language. Although 6 participants (Participants 1, 3, 4, 6, 7 and 8) reported using glasses daily, all participants opted for not wearing glasses during the experiment. Only

Participant 3 had contact with an eye-tracker before the experiment (for less than one hour). The participants were recruited through word of mouth in our social cycle and through a group of interest in a social network. To carry out the experiments, the participants had the same financial incentive reported by [Pedrosa *et al.* \(2015\)](#) (\$10 per session and \$25 for finishing all sessions), except that we did not offer a bonus reward for achieving the highest typing speed. In addition to running in real time, this experiment differs from the experiments reported in the previous chapters with respect to the adoption of the dictionary used by [Pedrosa *et al.* \(2015\)](#) to tune the ranking algorithm. We used it to provide a fairer comparison between them. In addition, all participants in this experiment have a daily contact with the English language whereas the logs used in our previous experiments were obtained through experiments with participants not as fluent in the language (c.f. Section 3.3.2).

Each participant engaged in six sessions, lasting from 45 to 60 minutes. We opted for using 6 sessions to investigate how the performance of the participants evolved as they used the system. During each session, the participants performed two blocks of typing with duration of 20 minutes, one per condition. Among the participants, four were taken at random to start typing with FilterTypedping in the first 20-minutes block (Participants 1, 3, 6 and 7) while the remaining four participants started with C-AMMLSEP-LHD (Participants 2, 4, 5 and 8). The second block of typing started when the participants felt ready to restart typing so they could rest as much as they wanted and feel less the effects of tiredness. No interval took longer than 10 minutes.

The experiment simulated the situation in which a user types natural language phrases with the eyes as quickly and correctly as possible. As in the work reported by [Pedrosa *et al.* \(2015\)](#), to simulate a free entry while still maintaining a controlled environment, the system suggested random sentences from a collection of 500 phrases contributed by [MacKenzie & Soukoreff \(2003\)](#) to the participants. The amount of transcription sentences depended on how many sentences the user was able to type in a period of 20 minutes. Participants were not interrupted in the middle of a sentence in case the time had run out during the sentence.

In each session, the participants had access to the typing tool, the eye tracker¹ required to perform the experiment, a chin rest, earphones, a 21.5 inches monitor and NASA Task Load Index (NASA-TLX) questionnaires². Before each block, the participants were instructed to comfortably accommodate themselves looking at the monitor at a distance of 60 cm and positioning their heads on a chin rest. They were free to choose the best height of the chair, chin rest and table. Figure 24 shows the setup of the environment. The experiment took place in a room located in the premises of the Dynamic Graphics Project Laboratory (DGP) at the University of Toronto. A calibration procedure was performed before each block of typing to reduce noise caused by the eye-tracker. The calibration of the eye-tracker consisted of observing 9 dots shown on the screen for about 2 seconds each. The participants were free to perform this operation as many times as

¹ In this experiment we used an affordable 30 Hz device called EyeTribe.

² In our experiments we used an online form (Available at <http://nasatlx.com/>), and accessed from November 28 to December 6, 2017) that was filled up after every block.

they wanted until they were satisfied with the result.

Figure 24 – Environment setup

In the first session, the participants were introduced to the procedures of the experiment and detailed information on how to use the interface. For training purposes, participants had the opportunity to freely use the interface in a test block until they felt safe to begin the experiment. The test block allowed participants to familiarize themselves with the system and eliminate questions regarding the interface. This block was applied only in the first session and it was discarded of our analysis.

During the rest interval between the two blocks, and at the end of the session, the participants filled in the NASA-TLX form according to their experience. The researcher also asked them to discuss the strategies they used while eye-typing and any problems they encountered, and collected their feedback regarding the design of the interface. Participants were not informed about the differences between the two blocks in order to find out if they were able to notice them.

In order to control the interference of motivational variables to perform tasks, such as fatigue, the participants performed a maximum of two sessions on the same day, with a minimum interval of 2 hours between sessions. Also, no more than 72 hours passed between each session.

6.2 Interface

Figure 25 shows the main screen of the Filterypedping interface we used, which corresponds to the one reported in the first phase of the study by [Pedrosa et al. \(2015\)](#). The interface suggests up to 6 words per page (bottom) and provides audio feedback for confirmed words, page change and for the *Delete word*, *Backspace* and *Enter* buttons. In the layout, all buttons are spaced from each other to facilitate action recognition. The sequence of looked-at letters was reset whenever the participant looked outside the keyboard.

During the eye-typing blocks, the system provides users with short phrases to be transcribed one at a time. To enter each word of a phrase, users have to move their eyes on a QWERTY keyboard according to the sequence of letters belonging to the desired word. If the

Figure 25 – Filtertypedping interface. Interface *Top left* is shown when the user looks below the QWERTY keyboard. Interface *Bottom left* is shown when the user looks to the right of the QWERTY keyboard. *Top left*: QWERTY keyboard on center, with key looked at highlighted (only when user waits for the feedback); sentence to be type on top left; word typed (and selected) below sentence to be typed. *Top right*: (optional) sequence of letters looked-at with the use of dwell (left), six suggested words (center), button for next six suggestions (right). *Bottom left*: Options for deleting one word and one char at a time, and for accepting the phrase as shown below the sentence to be typed on the top.

user looks at one key long enough, the gazed key is highlighted to ensure the user that the system detected the key she was looking at.³

To indicate that she had finished writing the word, the user looks at the area below the keyboard. Once the user gazes at the area below the QWERTY keyboard, the interface shows a set of (up to) six word suggestions. The user then gazes at each word in sequence until finding the desired one.

Note in Figure 25 (*top right*) that the first and last buttons at the bottom of the screen are not part of the set of suggestions. The first button shows the sequence of letters detected by the system when the user opted to type with a dwell time. The last button allows the participant to request a new list of suggestions if none of the words presented is the one requested by her (*forward* button →). Although the dwell typing option is available, we asked the participants not to use it.⁴ The *Dwell* button is only available at the first page. At the following pages, it is turned into a *backward* button (←).

To select a word, the user looks at the corresponding button and then look back at the virtual keyboard. She also has the option of restarting the writing process instead of choosing one of the words listed. To accomplish this, the user gazes at the *backward* or *forward* buttons

³ The eye fixation time for the feedback to be given was set up as 500 ms in our experiment.

⁴ The dwell time was set to 1 second.

and then at the virtual keyboard.

When a user types a wrong word, she can look at the button located at the upper right corner of the virtual keyboard to request a list of buttons that allows her to delete the entire word (*Delete*), delete the last letter (*Backspace*) or indicate to the system that she finished the transcription of the phrase (*Enter*). To select a command, the user should gaze at the desired button and gaze back at the virtual keyboard. After deleting the misspelled word, the user can rewrite the word. When the user finish all the words in a sentence, she has to select the *Enter* button. The system answers to this action by showing the obtained typing rate, which stimulates the user to improve her performance.

The program also allows the user to change characters from lowercase to uppercase, and *vice versa*, or to use a keyboard of special characters. These options are shown when the user looks at the upper left button in the virtual keyboard area. Since the sentences to be transcribed during the experiments had only lowercase letters, with no punctuation nor special symbols, the participants were instructed not to use these options.

More details about the interface are given in the work by [Pedrosa et al. \(2015\)](#).

6.3 Collected data

We registered all the interaction data collected over the sessions, including the detected letters and lists of words suggested to the participants. All data extracted were anonymised and made accessible only to the researchers. The letters detected were processed by the Kumar smothing algorithhm (cf. Section 2.6) as described by [Pedrosa et al. \(2015\)](#).

We discarded sentences that were reported as mistaken by the participants (Filteryedping: 26 sentences; C-AMMLSEP-LHD: 18 sentences). The sentences in which the participant requested the uppercase keyboard were also discarded since that action was interpreted as a mistake (20 sentences; 10 of each condition). The final number of sentences entered by the participants was 1677 (Filteryedping) and 1910 (C-AMMLSEP-LHD). Table 10 shows the number of sentences suggested by the system, the number of words in these sentences, the number of words in the sentences transcribed by the participants, the number of words selected by the participants including deletions, the total number of letters typed and the average length of the words typed by participants.⁵ These last two numbers include all the words participants selected, regardless of whether or not they were the intended words. In our study, we considered all the words the participants selected as the typed words.

⁵ Note that the transcribed sentence can be different from the sentence suggested by the program because users can finish typing the sentence any time they want. As a result, the number of transcribed words may be different from the number of all words the participants should type. Similarly, the number of selected words can be different from both since the users can select a word and delete it later. In practice, the number of words selected will always be greater than or equal to the number of transcribed words.

Table 10 – General statistics for Filteryedping and C-AMMLSEP-LHD.

	Filteryedping	C-AMMLSEP-LHD
Number of sentences	1,677	1,910
Number of suggested words	9,011	10,291
Number of transcribed words	8,984	10,255
Number of selected words	10,280	11,175
Number of typed letters	45,213	50,107
Length of words in sentences (avg)	4.398	4.483

6.4 Metrics

We used the WPM (words per minute) metric to measure the entry rate of each sentence (c.f. Section 2.7). This metric is commonly used to evaluate the typing rate in transcription tasks, as it is the case for the approaches reported by [Bee & Andre \(2008\)](#), [Sarcar, Panwar & Chakraborty \(2013\)](#) and [Pedrosa *et al.* \(2015\)](#). The final rate was obtained as the average over all sentences. The duration of each sentence was counted from the moment the participant started to type the first word to the moment she finished to type the last word. The time spent reading the sentence and hitting the *Enter* button was not included in our analysis. Since the words typed with the methods were randomly selected, it was expected that there would be differences between the average size of the words typed in each metric, as we can see in Table 10. To correct this bias, we considered words as sequences of 5 letters including spaces as in the work reported by [Pedrosa *et al.* \(2015\)](#).

The word counting to estimate WPM includes all words the participants selected since every time they select a word and delete it later the entry rate is affected with consequent performance decreasing. The number of words selected by the participants outnumbered the transcribed ones in 12.6% when using Filteryedping and 8.2% when using C-AMMLSEP-LHD. The participants selected additional words when (i) they requested the suggestion list by mistake or (ii) they did not find the word they wanted and selected any word in an attempt to return to the typing process. Since all of these actions were not really desired by the participants, we applied a penalty based on the extra selected words that we called Incorrect Fixed Rate (IFX). The final sentence transcribed by a participant can differ from the suggested one, so we still applied a penalty according to the Minimum String Distance (MSD) existing between the transcribed sentence and the sentence provided by the program. This penalty is the Incorrect not Fixed Rate (INFX) and the combination of the two penalties is the Overall Error Rate (OER). We calculated OER as follows:

$$OER = \frac{INFX(T, P) + IFX(S, P)}{C(T, P) + INFX(T, P) + IFX(S, F)}$$

Where *INFX* is the minimum amount of operations to turn the transcribed sentence *T* into the suggested sentence *P*; *IFX* is the character difference between all selected words *S* and

P , including spaces; and C is the amount of matching characters between the transcribed phrase and the suggested one. The final WPM is obtained as follows:

$$WPM = \frac{\text{Number of times a word was selected}}{\text{Time the participant spent to type the sentence}} \times (1 - OER)$$

If the participant does not find the correct word in the suggestion list, the interface allows her to retype it without selecting some suggestion. However, we observed that the participants preferred to select a word to delete it later. Since the number of times a word need to be retyped affects the WPM, we adopted as *Deletion rate* the percentage of deleted words in the set of selected words.

Besides retyping, another factor that influences the typing performance is the position a word is placed in the ranking. As before, we used MRR to compare the generated rankings since it favors methods that rank correct words at the top. Note that, in actual cases, the users frequently give up the search before visiting the page on which the correct word is. Therefore, differently from our previous offline evaluation, the MRR values in these experiments can be often affected by the user's willingness to search for the correct word.

To evaluate the task workload for each condition, we used the NASA TLX Assessment tool in which participants gave scores and weights for Performance, Effort, Frustration, Mental, Physical and Temporal demands. The session score was taken as the average of each weighted factor while the final score was taken as the average of all sessions.

Like our previous experiments, for all the reported results obtained from an average, we also provided confidence intervals considering a 95% confidence level.

6.5 Results

6.5.1 Entry rate

Overall C-AMPH-LHD improved eye-typing performance in all sessions,⁶ with participants achieving 19.12 ± 0.438 WPM in average in the last session, in which the gain over Filtertypedping was about 12%. When considering all sessions, the participants typed faster using C-AMPH-LHD (16.87 ± 0.22 WPM) than using Filtertypedping (15.17 ± 0.24 WPM), with a gain of 11.2%. Figure 26 shows the entry rate per condition per session. The biggest gains over Filtertypedping were in the first and second sessions (21.4% and 21.0%, respectively). These results suggest that a method for reducing noise can be particularly useful when the user is more inclined to make mistakes, such as when she is not used to the interface or does not use it often. In these situations, she is more likely to perform actions which negatively affects her intended typing.

⁶ Except in Session 3 which, however, presented a p-value < 0.05 when we applied a paired t-test. As observed, this suggests that the confidence intervals will not overlap for a larger sample size.

Figure 26 – WPM per session.

Figure 27 – WPM per participants.

Figure 27 shows the entry rate per condition for each participant. Participants 1, 3, 6 and 7 started typing using Filteryedping then moved to C-AMPH-LHD, while participants 2, 4, 5 and 8 did the opposite. We did not find any notable differences between the two groups when analyzing their overall performance. C-AMPH-LHD achieved significant gains for all participants, except by Participants 3 and 8. On average, Participant 3 was able to perform well using both methods (1.22 WPM above the mean when using C-AMPH-LDH and 2.74 WPM when using Filteryedping). Participant 6 was the typist who obtained the largest gain with C-AMPH-LHD (28.39% faster than when using Filteryedping). The fastest typist was Participant 7, using C-AMPH-LHD, who achieved on average 19.91 ± 0.52 WPM. Using Filteryedping, Participant 3 was the fastest one achieving 17.91 ± 0.61 WPM on average.

Figure 28 compares the performance of the participants in the first and last sessions. As expected, in the last session all participants were on average better than in the first session using both methods. Since in the first session the participants have no previous contact with the tool, it is expected that they perform better in their second block which means that they should be faster using the second block's typing method. This was exactly which we observed for all participants who started using Filteryedping. They were faster after switching to C-AMPH-LHD.⁷ However, among the ones who started typing with C-AMPH-LHD, none of the participants had significant gains. Participants 2 and 4 were on average faster after switching to Filteryedping while Participants 5 and 8 were not. This is probably due to the fact that C-AMPH-LHD is more robust to noise than Filteryedping. As result, when users switch to Filteryedping, they have more difficult to adapt, even after familiarized with the interface. The highest speed at the first session using C-AMPH-LHD was 17.83 ± 1.70 WPM (Participant 7), while for Filteryedping it was 14.57 ± 1.64 WPM (Participant 2).

By the time participants started the last session, each participant had already performed more than 1 and a half hours of typing with both techniques, so we expected that the typing order of each method would not be a decisive influence factor in the results. However, Participants 2,

⁷ In fact, confidence intervals for Participant 3 overlap. However, a paired t-test resulted in a p-value < 0.05 , which indicates non-overlapping intervals for a larger sample size.

4, 6, 8 and 5 were significantly better with C-AMPH-LHD. Due to its robustness to noise, C-AMPH-LHD is a technique that provides easier typing for users when compared to Filteryedping. Similarly to the observed in the first session, these results show that it was easier for the participants making the transition from a difficult to an easier technique than the opposite. In this session, the fastest typists were Participant 4 (21.40 ± 1.06 WPM) when using C-AMPH-LHD, and Participant 7 (20.53 ± 1.06 WPM) when using Filteryedping.

Figure 28 – WPM per participants for Sessions 1 and 6.

Figure 29 – Deletions per session.

6.5.2 Error rates

We used the average minimum string distance (MSD) error rate to compare the amount of errors existing in the transcribed sentences. The methods presented a small MSD rate (C-AMPH-LHD: 0.8 ± 0.2 ; Filteryedping: 0.9 ± 0.2) with no significant differences between them. Similarly, comparisons between sessions did not bring up differences. This was probably due to the use of the same interface and procedures in all cases. In fact, the sentences chosen for the task, how they are presented, and the opportunities user had to check and correct their errors are likely to be more related to the chances of a participant making a mistake than to the position in which the words appear in the ranking.

We observed that most of the time participants did not find the word they wanted, they preferred to select any word and delete it later. The amount of times they retyped without selecting any options of the list was small for both conditions (C-AMPH-LHD: 64 times; Filteryedping: 80 times). Figure 29 shows the percentage of deletions for each session per condition. In general, participants made fewer deletions when using C-AMPH-LHD. For both methods, the percentage of deletions decreased as the participants acquired experience. Similarly to the observed with respect to the entry rate (cf. Section 6.5.1), the difference between the methods is larger in the first two sessions, when the users had no experience with the interface.

6.5.3 Ranking evaluation

Table 11 shows the percentage of correctly ranked words suggested by each condition according to their positions. Taking into account the number of pages that the participants effectively walked through, the MRR rate for C-AMPH-LHD was 0.807, which represents a gain of 12.46% over Filteryedping (MRR = 0.717). In particular, C-AMPH-LHD ranks the words better mainly at the first position. Using C-AMPH-LHD, the participants missed 40% fewer words (6.41% of words not found) than when using Filteryedping (11.34%). For both methods there is no expressive increase of suggested words after the second page. These results are quite similar to those observed in our last experiments (cf. Table 9) in spite of their difference on the native language of the users, the adopted dictionary, and the nature of the analysis (online versus offline).

Table 11 – Proportion of words found in Top-1, 6, 12, 18 positions. “All” stands for all suggestions found.

method	Top-1	Top-6	Top-12	Top-18	Top-all
C-AMPH-LHD	74.24 $\pm$ 0.81	90.50 $\pm$ 0.54	92.48 $\pm$ 0.48	93.19 $\pm$ 0.46	93.59 $\pm$ 0.45
Filteryedping	61.49 $\pm$ 0.94	86.43 $\pm$ 0.66	88.15 $\pm$ 0.62	88.49 $\pm$ 0.61	88.65 $\pm$ 0.61

6.5.4 NASA Task Load Assessment

Users were not aware of the difference between the two typing blocks. We applied NASA TLX to find out the required workload for each condition and whether participants reported differences between them. Overall, we did not find significant differences between Filteryedping and C-AMPH-LHD (C-AMPH-LHD: 41.59 $\pm$ 5.39; Filteryedping: 48.22 $\pm$ 5.17). Figure 30 shows the final score for each session. Although no significant differences are observed, the result suggests a workload decrease as sessions progress, since the confidence intervals are narrower as the workloads are smaller. Figure 31 shows the final score for each scale. As before, no significant differences were observed. From these results, we can conclude that participants did not perceive the effects of the ranking algorithms in terms of workload nor regarding performance. For them, the methods are generally similar, which may be related to the fact that most of users experience is associated with elements shared by the methods as discussed above. Thus, the list of suggestions represents only a small portion of the complete user experience.

6.5.5 Participant feedback

In addition to filling in the NASA TLX tool, participants were asked if they had any problems while typing and if they felt any difference between the two blocks. The major problem reported was the calibration of the device. The main observations made by the participants are listed below:

- Regarding physical discomfort, Participants 1 and 3 complained about the brightness of the screen; Participant 2 complained about the distance of the screen to her eyes; and

Figure 30 – NASA TLX scores for Sessions 1 and 6.

Figure 31 – Final scores per scale for each condition. Me, Ph, Te, Pe Ef and Fr stand for Mental demand, Physical demand, Temporal demand, Perform, Effort and Frustration scales respectively.

Participant 8 did not like to look at the screen for a long time. Participants 7 and 5 reported fatigue when typing due to the time duration of each block;

- Participant 1 stated that she was not interested in knowing her speed and did not pay attention to the speed reported on the screen after finishing the sentence. Lack of interest in speed can lead a participant not to reach her best performance;
- Some participants (Participant 1, Session 5; Participant 8, Session 2) felt sleepy during the execution of the typing tasks which may also compromise their performance;
- Participant 1 and 5 think that C-AMMLSEP-LHD is more accurate for words in which the letters are in farthest positions on the keyboard;
- Participants 1 and 6 noticed that the calibration gets worse during the typing block although they did not know the reason.
- Participant 5 reported problems typing double letters using C-AMMLSEP-LHD.

Most of the time participants did not notice differences between the two methods. However, when they noticed differences, these were related to the ranking of words. Participant 2 (Session 5), Participant 4 (Sessions 3, 4, 5 and 6) and Participant 5 (Sessions 5 and 6) reported that the correct words appeared better ranked during the blocks they typed using C-AMMLSEP-LHD. Participant 4 observed that Filteryedping suggested words much different from what one would expect such as “zzz” or “aaaa”. Participant 5 said that she felt faster when using C-AMMLSEP-LHD because the correct word appeared in a good position on the suggestion list even when she did not gaze at all the letters she should have gazed. Participant 2 said she felt less need to retype words when using C-AMMLSEP-LHD. Participant 8 said her performance was always better using C-AMMLSEP-LHD but she did not know the reason. All participants

belonging to the group that typed using Filterypedping in the first block and C-AMMLSEP-LHD in the second one (Participants 1, 3, 6 and 7) did not report any difference between the two methods. In spite of the small number of participants, the fact that all participants in the group who used C-AMMLSEP-LHD in the first block noticed differences between methods reinforces the idea that C-AMMLSEP-LHD might be more useful when the user initiates their contact with the program and therefore produces more noise. The observations made by Participants 2, 4 and 5 will be examined in next subsection.

6.5.6 Interface-related actions

We now evaluate how the changes in Filterypedping affected the actions performed by the participants during typing. We divide the process of typing a word into three subprocesses: Writing, Selection and Deletion. The proportion of time spent in each action is presented in Table 12. We consider that the *Writing time* of a word begins when the participant starts gazing at the keyboard until the moment she looks at another region. The *Selection time* comprises all the time the participant has been looking at the region where the suggestions are presented. During this time, the participants performed the actions of reading the suggestions, walking through the buttons and pages and choosing a suggestion. The *Deletion time* comprises the time taken to perform the deletion action, i.e., the time elapsed since activating the delete options menu (*delete word* and *delete char*) by looking at the rightmost portion of the interface until selecting the desired deletion button.

Table 12 – Time spent in each action per word (%).

	All words			Deleted words		
	Writing	Selection	Deletion	Writing	Selection	Deletion
Filterypedping	65.76 $\pm$ 0.44	31.95 $\pm$ 0.41	2.28 $\pm$ 0.17	45.18 $\pm$ 1.24	35.91 $\pm$ 1.19	18.89 $\pm$ 1.12
C-AMPH-LHD	66.65 $\pm$ 0.45	31.61 $\pm$ 0.42	1.72 $\pm$ 0.15	43.01 $\pm$ 1.21	37.43 $\pm$ 1.50	19.55 $\pm$ 1.10

As shown in Table 12, the distribution of the time spent in each action per word was similar for both conditions. As expected, the most time-consuming action was *Writing*. However, it is important to notice that *Selection time* took almost half of the *Writing time*, which indicates that alternative approaches for this action may significantly improve typing speed. Deletion had little impact (less than 3% of the total time) when we consider all words because it was rarely performed. Only 12.56% of the words were deleted using Filterypedping while 8.21% were deleted using C-AMPH-LHD. Moreover, the action of pressing the button is too short for having an important impact (the time participants spent to press the *Deletion* button was 1178.21 $\pm$ 55.40 ms). However, the impact of the action is relevant when we consider just the words that were deleted. For these words, more than 15% of the typing time was spent in the deletion action. The difference between the time spent selecting a word when compared to writing it is also important to be observed. Usually deleted words occurred when participants did not find the words they were looking for in the suggested list (77.97% of the deletions were cases in which

C-AMPH-LHD did not suggested the intended word. For Filteryedping, this number is 90.32%). Thus, they spent more time looking for the words in the list than they would usually spend.

In the following, we discuss the effect of each action in more details.

6.5.6.1 Deletion

Changing the ranking algorithm affects deletion since when the user does not find the word she wants in the suggestion list, she usually selects any of the presented words only to delete it immediately, as a means of restarting the writing process. Table 12 shows that the time it takes to perform the deletion action itself does not have much impact on the time it takes to type a sentence. However if we consider all the consequent actions (unsuccessfully looking for a word, selecting and deleting another word), we will find that the deletion action can lead to a significant waste of time and the less often the users need to perform it, the higher will be their speed. Table 13 shows how much time the participants spent on average typing and selecting correct words and correcting the wrong ones when typing a phrase. As we can see, C-AMPH-LHD spends almost 40% less time correcting words than Filteryedping.

Table 13 – Time spent in each action in average per word (ms).

	Writing	Selection	Correction
Filteryedping	15,439.760 $\pm$ 235.398	7,071.873 $\pm$ 148.670	5,975.994 $\pm$ 627.520
C-AMPH-LHD	14,404.276 $\pm$ 190.193	6,558.991 $\pm$ 139.175	3,564.428 $\pm$ 323.460

6.5.6.2 Writing

The method used to suggest words can also influence the time a user spend writing. The writing time is directly influenced by the number of letters users gaze until they ask for the suggestions menu. Since C-AMPH-LHD can recognize words corrupted by different kinds of errors besides deletions, it does not require users to look at all letters of the word. As a consequence, users spent less time typing, on average, when using C-AMPH-LHD (2744.37 $\pm$ 33.03 ms) than when using Filteryedping (3001.20 $\pm$ 41.48 ms, loss of 9.35%). Figure 33 shows the percentage of words the participants typed without looking at all letters for each condition. Even though Filteryedping does not deal with missing letters, the method works around this problem by adding neighbor letters to the input string when it detects a user gaze in an overlapping key region. According to the results, C-AMPH-LHD was more effective in returning the correct word for words typed with missed letters. Moreover, its users were more likely to take the risk to skip letters. The gains associated with missing letters was larger in the firsts two sessions, which reinforces the idea that C-AMPH-LHD is more useful in noisier situations.

6.5.6.3 Selection

This is the action more affected by an effective ranking: the better the ranking, the shorter is the time spent looking for the correct word. As seen in Table 11, C-AMMLSEP ranks the correct

Table 14 – Pages visited by participants until selecting a word.

Visited pages	C-AMPH-LHD					Filteryedping				
	Mean Time	± CI	N. Words	% Words	± CI	Mean Time	± CI	N. Words	% Words	± CI
0	1,760.55	242.64	22	0.19	0.08	-	-	-	-	-
1	1,118.30	12.96	10,486	94.07	0.44	1,253.26	14.87	9593	93.27	0.48
2	3,904.24	119.24	360	3.23	0.33	3,905.77	105.86	438	4.26	0.39
3	5,862.53	206.59	161	1.44	0.22	5,412.37	224.01	161	1.57	0.24
4	7,360.35	290.57	84	0.75	0.16	7,077.88	509.28	57	0.55	0.14
5	9,443.26	850.00	27	0.24	0.09	8,931.72	827.55	25	0.24	0.10
6	11,757.19	1,262.75	16	0.09	0.06	11,611.00	2,981.17	5	0.05	0.04
7	13,160.40	3,725.28	10	0.09	0.06	11,743.80	3,322.02	5	0.05	0.04
8	12,734.33	11,848.23	3	0.03	0.03	14,861.00	0.00	1	0.01	0.01

word better than Filteryedping, which translates into 10.76% less time spent in selection (1118.27 ± 25.26 ms for C-AMPH-LHD and 1253.26 ± 24.47 ms for Filteryedping, on average).

Table 14 shows the maximum number of pages checked by the participants when searching for a word in the list of suggestions. The number of visited pages is 0 when the method returned an empty list of suggestions. This happened to C-AMPH-LHD a total of 22 times. That happens because C-AMPH-LHD is limited to dictionary words whose edit distance corresponds up to 10 deletions and 2 other operations. Thus, if the input sent to C-AMPH-LHD is very different from any dictionary word, C-AMPH-LHD may not find a reasonable candidate in the dictionary. Differently from C-AMPH-LHD, Filteryedping always returns suggestions even if they are unlikely. Comparing the results from Table 14 and Table 11, we observe that the number of times the participant stopped searching for the word on the first page was higher due to the fact that both methods returned a large amount of correct words in the six positions of the first page (Filteryedping: $86.43 \pm 0.66\%$; C-AMPH-LHD: $90.57 \pm 0.54\%$). However, the number of times the participants stopped searching on the first page was greater than the number of times the methods ranked the correct keywords in the first 6 positions, which suggests that a user may give up searching for a correct word if it is not in the first page.

The time spent on each page may be influenced by whether or not the user has found the word in the list of suggestions. When she finds it, the time is determined by the position where the word is. If she does not find, the time is determined by her willingness to find it. Our experiment showed that C-AMPH-LHD takes more time when a user does not find the typed word in the suggestion list (participants spent in average 3487.06 ± 210.58 ms when using C-AMPH-LHD and 2872.37 ± 127.33 ms when using Filteryedping). To better justify this discussion, Figure 32 presents the maximum number of pages walked by the participants when the correct word was not in the list of suggestions for each condition. As observed, the participants were more likely to stop searching on the first and second pages while using Filteryedping whereas they looked into more pages while using C-AMPH-LHD. This is probably due to the fact that C-AMPH-LHD suggestions encourage the user to walk through more pages looking for a candidate. As it presents a more consistent ranking, it leads the user to believe that the correct word may be present and it is worth looking for it. To illustrate that, Table 15 shows examples of 5 random words that were

Table 15 – Examples of suggestion lists associated with typed words which were not found.

	Correct word	Options	Max. pages walked
C-AMPH-LHD	knee	june;keene;jane;ken;like;kent;keen;kevin;one;g...	4
	was	as;do;dow;is;so;dsw;day;sw;did;few;saw;dew;w;d...	3
	zoom	do;from;doom;dom;of;room;on;door;to;tom;don;or...	2
	doors	does;doris;more;doses;done;sores;cores;dores;d...	3
	offer	of;to;or;for;roger;tour	1
Filteryedping	rejection	reject;room;sheet;she;seen;soon	1
	team	the;it;item;i;iii;oh;ieeee;tee;hmm;ii;ooh;mmm	2
	cotton	in;on;orion;iron;corn;xxxiii;orin;or;xxiii;co...	2
	heroic	of;to;off;too;roof;if	1
	experience	pepper;power;were;force;prince;price;piece;for...	4

not found by participants and the suggestions they saw until they gave up for each condition.

Figure 32 – Percentage of pages visited by the participants when the correct word was not found.

Figure 33 – Percentage of words participants did not gaze to all letters per session.

6.6 Final Considerations

In this chapter, we evaluated our method by comparing it with a state-of-the-art eye typing input system, Filteryedping. In particular, we ran a user experiment with 8 participants which performed 2 hours of typing, divided into 6 sessions, using both methods. While it is not possible to state that users clearly perceived advantages in using C-AMPH-LHD as compared to Filteryedping, they were significantly faster. More specifically, they achieved 17.46 WPM in average, when considering all sessions, and 19.80 WPM in the last session, which represents a gain of 11.3% over Filteryedping. The system was more useful in the first sessions when the participants were still not used to the interface and the input was noisier.

We also observed that the fastest typists were the ones that took more risks while typing. They usually skip letters and rely on the ability of C-AMPH-LHD to recognize the resulting words anyway. These results are important because these systems have to be used by physically disabled people. As their illness progresses, more noise is expected in the produced input. Regarding the ranking quality, C-AMPH-LHD was able to rank 17% more words in the first position than Filteryedping and missed 40% less words, leading to a gain of more than 12% in

MRR. Finally we analyzed the impacts of the ranking change in each typing action. The new model was able to decrease the time spent in all evaluated actions. Besides, probably as result of more consistent rankings, we observed that the users were more willing to search for the correct word among the suggestions provided by C-AMPH-LHD than the ones provided by Filteredping.

As with the previous chapter, the results reported here are part of the journal article “A Noisy-channel based Model to Recognize Words in Dwell-free Eye Typing Systems”, currently under review by the ACM Transactions on Computer-Human Interaction.

CONCLUSIONS AND FUTURE WORK

In this work we have proposed a spelling correction method for recognizing words typed with the eyes. Our solution comprises two main components: an approximate search method to select candidates and a ranking algorithm to order the candidates so that the most likely word is the first one in the suggestion list.

The search component consists of an asymmetric Mor-Fraenkel approach where d - and k -residual strings are matched, with d likely larger than k . We have evaluated our method using, as search structures, a Trie and a minimum perfect hashing. The search component was built to support the pruning of the processing so that the response time is guaranteed to be no longer than a certain predetermined limit.

The ranking component adopts a noisy-channel based model to find the most probable words. To accomplish this, it combines a language and noise components. In turn, the noise model combines general and specific domain knowledge about errors. In particular, the edit probabilities were derived from a general error corpus and from specific error source knowledge. Using such information, two different algorithms were proposed to estimate the likelihood of a set of edit operations, given an input pattern:

- A quadratic exact approach based on the maximum message length optimum alignment
- A heuristic approximation which benefits from the processing order of the residuals and information stored in the Mor-Fraenkel index. In particular, the ordered processing allows that the more promising candidates are evaluated earlier, enabling prune strategies based on limiting processing time.

We also provide comprehensive studies on (i) how to balance the influence of the language and noise components of the noisy-channel based model; (ii) how to include information associated with the eye movements to better capture errors related to movement variability; and (iii) the impact of preprocessing the input stream to filter unintended selections made during fast eye movements and diminish the jitter characteristic of inexpensive eye tracker devices.

We evaluated our models using real log interaction data collected with a real eye typing system provided by [Pedrosa *et al.* \(2015\)](#). This data enabled us to tune our methods so that we evaluated our best approach in a real transcription experiment using an adapted version of the eye typing system contributed by [Pedrosa *et al.* \(2015\)](#). Based on our offline evaluations, our findings are as follows:

- A noise model based on specific knowledge about eye-typing error outperformed other approaches based on general errors. Its performance was even better when using specific edit probabilities estimated by combining information from a general corpus and the eye typing domain. This method was also more robust to variations of the dictionary size. Its high precision also contributed to minimize the need for context changes in an eye typing interface.
- Our Mor-Fraenkel method combined with a minimum perfect hashing largely outperformed the Trie version regarding time efficiency. In particular, the best performance in terms of response time was obtained by the linear version (AMPH-LHD) which was just slightly inferior to the MML-based version (AMPH-MML) in terms of ranking quality (MRR).
- When taken in isolation, strategies such as balancing noise and language components, input pre-processing, and contextual probabilities have little impact on word recognition ability. However, when combined, they enabled our last model (C-AMPH-LHD) to significantly outperforms our previous best approach (AMPH-LHD) with a gain of 10.16% in MRR.

Regarding the online evaluation, we compared our best method, C-AMPH-LHD, with a state-of-the-art method, *Filtaryedping*. We ran a user experiment with 8 participants which performed 2 hours of typing, divided into 6 sessions, using both methods. While it is not possible to state that users clearly perceived advantages in using C-AMPH-LHD as compared to *Filtaryedping*, they were significantly faster using C-AMPH-LHD. More specifically, they achieved 17.46 WPM in average, when considering all sessions, and 19.80 WPM in the last session, which represents a gain of 11.3% over *Filtaryedping*. The system proved be more useful in the firsts sessions when the participants were still not used to the interface and the input is noisier.

We also observed that the fastest typists were the ones that took more risks while typing. They usually skip letters and rely on the ability of C-AMPH-LHD to recognize the resulting words anyway. These results are especially important because these systems have to be used by physically disabled people. As their illness progresses, it is more difficult for them to hit the keyboard letters and therefore, more noise is expected in the produced input. Regarding the ranking quality, C-AMPH-LHD was able to rank 17% more words in the first position than *Filtaryedping* and missed 40% less words, leading to a gain of more than 12% in MRR. Finally we analyzed the impacts of the ranking change in each typing action. The new model proved be able to decrease the time spent in all evaluated actions. Besides, we observed that the users were more willing to search for the correct word among the suggestions provided by C-AMPH-LHD

than the ones provided by Filteredping — this may be due to the offering of more consistent rankings by C-AMPH-LHD.

7.1 Hypothesis an research questions revisited

The following hypothesis, formulated in Chapter 1, guided the research carried out in this thesis:

Hypothesis: *A ranking based on a noise model that takes advantage of the typical noise in eye typing can improve user performance and provide a better user experience. Although this ranking is more complex and therefore has a greater computational cost, it is possible to implement it in a viable way by taking advantage of the distribution of errors observed in eye typing.*

The hypothesis lead to the formulation of the research questions discussed next.

How could the noises specific to the eye-based typing domain, in particular when considering affordable eye trackers, be modeled to improve word recognition?

We have modelled specific errors by associating them with four edit operations: insertion, deletion, substitution, and transposition. With each operation we associate a probability so that we can estimate how likely a set of operations is. Given two strings s and t , this information can be used to calculate the likelihood of s to be transformed into t . This comprises a noisy channel-based model which considers that the similarity between the corrupted signal and the correct information is proportional to that likelihood.

Among the specific sources we have modeled, the most important was the eye tracker device — which is of particular importance when low cost devices are used. As observed by Beers (2007), inexpensive devices can be important sources of error, which we confirmed in practice. As for the other errors we modeled, derived from eye movements variability and erratic user behaviors, its inclusion resulted just in a slightly improvement on precision. In addition to using a chin support, we did nothing else to avoid calibration problems. These were the largest source of noise, in our experiments, which were not explicitly captured by the models.

How to combine eye-based typing specific noises with general noise observed in any typing system to design an effective word recognition model?

We combined the different noise-related information using a simple linear combination where parameter α is used to weight the general corpus evidence. For non-contextual probabilities, best α is around 0.7. For contextual probabilities, when the origin of saccades is taken into account, best α is close to zero, mainly due to the large number of events to count. This suggests that a combination in log space should have been attempted to explore the parameter space. To

balance the influence of the error against the language model, we used a linear combination in log space, controlled by parameter λ . In that case, we observed that the best value for non-contextual probabilities is $\lambda = 2$ and for non-contextual probabilities, $\lambda = 2.4$. We solely evaluated the unigram model, leaving alternative models for future work.

How could these models be efficiently estimated, in real time, so that they are feasible in a real interface?

Even response times around 200 milliseconds will hardly be noticed by the user, since this is the average eye fixation time. However, since the input may be only one among the many tasks running concomitantly on a computer, the desirable response time should be much shorter. A value commonly used in related work, and our target in this work, was somewhere around a few tens of milliseconds or less in a regular machine.

To estimate likelihoods, we first used a maximum message length (MML) approach which has cost proportional to the product of the length of the input string and the dictionary word. This estimate has to be made for all words of a large enough dictionary.

We minimized costs first by filtering out characters in the input stream using a mode filter. We then reduced the search space by selecting promising candidates. To accomplish this, we took advantage of the particular edit operation error distribution observed in the eye typing scenario to adapt a approximate search approach based on a Mor-Fraenkel index and a Trie. The resulting algorithm was able to provide acceptable average response time, though without ensuring a limit time to deliver a response. We then replaced the Trie with a minimum perfect hashing, proposed a linear heuristic to estimate the likelihoods using information already present in the Mor-Fraenkel index, and modified the search method to support cut processing with minimum loss in quality. The final algorithm achieved an average response time of about 10 milliseconds for a 70k dictionary. The MML-based version is just slightly slower. As consequence, in experiments with real users, they never noticed the processing time required to build the suggestion lists, using a regular computer, which we consider a satisfactory result.

What is the practical impact of these models on the user experience when evaluated in a real interface? Do they help the user to achieve faster typing speeds without losses in accuracy? Do they provide a more enjoyable use experience?

The users were 11% faster when considering all session, 12% when considering only the last sessions. These results were statistically significant in a 95% confidence level. Their accuracy was the same as those achieved using Filterypedping. As for their experience, we did not find statistically significant differences on the participants' perception about their use of both methods. This suggests that the gains of the ranking, although effective, were not large enough to be perceptible.

7.2 Limitations of this work

The following are limitations of the research carried out so far:

- Although the main motivation in this work is to provide an effective input device for people with disabilities, no experiments were carried out with subjects belonging to this population. Thus, we are not able to determine how helpful the results are for these users.
- Although time efficient, the space requirements of the proposed models can be prohibitive for memory constrained devices.
- The proposed models are not able to learn by themselves about user behaviors and preferences. This could be a problem for handicapped users whose motor capabilities decrease along the time.
- Although eye movement variability is an important error source, it was modeled based only on primary eye movements, i.e., saccades and fixations where the oculomotor muscles start in the rest position as observed by [Beers \(2007\)](#). Since scanpaths in eye typing include primary, secondary, and tertiary movements, more robust models have to be developed.
- Most methods proposed in literature are developed for native English speaking users. As consequence, reference collections and benchmarks require evaluation with these users. This naturally makes it harder to conduct experiments in Brazil — considering people with or without disabilities.
- It would be appropriate to carry out experiments using other eye tracker devices. Although many companies manufacture inexpensive eye trackers nowadays, none of them operates in Brazil. During the course of this work, we had problems to obtain devices, support, and software updates.

7.3 Future work

Along this work, many new questions arose. Also, some of our conclusions were conditioned to limitations in our methods and experiments. These issues have motivated new research to be pursued in the future, summarized in this section.

7.3.1 Improvements on Evaluation

We intend to carry out experiments with users with disabilities to better understand the impact of the noise model in such use scenario. One limitation of evaluation we performed is the lack of a spontaneous typing experiment, which impacts on the kind of errors committed by the user. We intend to improve this, in future, by observing subjects in spontaneous typing experiments such as the one proposed by [Baba & Suzuki \(2012\)](#), in which users were asked to describe images.

7.3.2 Improvements on Interface

Since these models are designed to be robust to noise, users that take more risk when typing (such as skipping letters) achieve higher speeds. In future, we intend to change the interface to stimulate such behavior by modifying the feedback strategies such that the users are induced to skip more letters.

We also intend to investigate how to adapt our model to predict the next letters during typing in such a way that we can shorten the dwell time according to the certainty that C-AMPH-LHD has about the next actions of the user.

7.3.3 Better prediction models

We intend to apply neural-based models to this problem, given their known robustness to noise and their natural ability to learn during use. A preliminary idea is to treat this problem as a translation problem. Bidirectional LSTMs with attention mechanisms have been successfully applied to this problem with outstanding results, as reported by [Luong, Pham & Manning \(2015\)](#). A possible difficulty with such models would be the availability of large amounts of parallel input to train the model. This, however, could be addressed by using syntetic data, generated with the help of eye movement models, such as the one proposed by [Beers \(2007\)](#).

7.3.4 Other issues

We intend to study alternatives to reduce space overhead, in particular, with the adoption of reduced alphabets and compressed perfect hashing as in the work by [Boytssov \(2012\)](#).

An issue with our proposed algorithms is the difficulty to update the dictionary, given the high cost of building a minimal perfect hashing. In future, we intend to improve this by using hierarchical minimal perfect hashing or other efficient strategies such as the one proposed by [Chegrane & Belazzougui \(2013\)](#). This work, in particular, is as fast and space efficient as our methods, but with faster building time. It has to be extended, however, to support transpositions.

BIBLIOGRAPHY

ALLISON, L.; WALLACE, C. S.; YEE, C. N. Finite-state Models in the Alignment of Macromolecules. **Journal of Molecular Evolution**, Springer, v. 35, n. 1, p. 77–89, 1992. ISSN 0022-2844. Disponível em: <<https://doi.org/10.1007/BF00160262>>. Cited 3 times on pages 21, 34, and 43.

ALLISON, L.; YEE, C. Minimum Message Length Encoding and the Comparison of Macromolecules. **Bulletin of Mathematical Biology**, Springer, v. 52, n. 3, p. 431–453, 1990. ISSN 0092-8240. Disponível em: <[https://doi.org/10.1016/S0092-8240\(05\)80219-8](https://doi.org/10.1016/S0092-8240(05)80219-8)>. Cited 3 times on pages 29, 34, and 40.

ANTONIO, N. S.; LEVY, P. C.; CAETANO, R.; SOUZA, P. G. Design of a Brazilian Portuguese Virtual Keyboard for People with Severe Motor Disability. In: **13th Brazilian Symposium on Human Factors in Computing Systems**. Porto Alegre, Brazil, Brazil: Sociedade Brasileira de Computação, 2014. (IHC '14), p. 14–20. ISBN 978-85-7669-291-1. Disponível em: <<http://dl.acm.org/citation.cfm?id=2738055.2738061>>. Cited on page 38.

BABA, Y.; SUZUKI, H. How Are Spelling Errors Generated and Corrected?: A Study of Corrected and Uncorrected Spelling Errors Using Keystroke Logs. In: **50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2 (ACL12)**. [s.n.], 2012. p. 373–377. Disponível em: <<http://dl.acm.org/citation.cfm?id=2390665.2390749>>. Cited 4 times on pages 22, 23, 29, and 103.

BASSIL, Y. Parallel Spell-Checking algorithm based on yahoo! N-Grams dataset. **International Journal of Research and Reviews in Computer Science (IJRRCS)**, v. 3, n. 1, fev. 2012. Disponível em: <<https://arxiv.org/abs/1204.0184>>. Cited on page 64.

BAYTIN, A.; GALINSKAYA, I.; PANINA, M.; SERDYUKOV, P. Speller performance prediction for query autocorrection. In: **22nd ACM International Conference on Information and Knowledge Management**. [s.n.], 2013. p. 1821–1824. ISBN 978-1-4503-2263-8. Disponível em: <<https://doi.org/10.1145/2505515.2507871>>. Cited on page 29.

BEE, N.; ANDRE, E. Writing with your eye: A dwell time free writing system adapted to the nature of human eye gaze. In: **4th IEEE tutorial and research workshop on Perception and Interactive Technologies for Speech-Based Systems**. Springer-Verlag, 2008. p. 111–122. Disponível em: <https://doi.org/10.1007/978-3-540-69369-7_13>. Cited 2 times on pages 38 and 88.

BEERS, R. J. v. The Sources of Variability in Saccadic Eye Movements. **The Journal of Neuroscience**, Society for Neuroscience, v. 27, n. 33, p. 8757–8770, 2007. ISSN 1529-2401. Disponível em: <<https://doi.org/10.1523/jneurosci.2311-07.2007>>. Cited 7 times on pages 23, 30, 69, 79, 101, 103, and 104.

BEERS, R. J. V. Saccadic eye movements minimize the consequences of motor noise. **PLoS ONE**, v. 3, n. 4, abr. 2008. Disponível em: <<https://doi.org/10.1371/journal.pone.0002070>>. Cited on page 69.

BOCEK, T.; HUNT, E.; STILLER, B. **Fast Similarity Search in Large Dictionaries**. [S.l.], 2007. Disponível em: <<http://fastss.csg.uzh.ch/>>. Cited 3 times on pages 37, 40, and 47.

BOTELHO, F. C.; PAGH, R.; ZIVIANI, N. **Simple and Space-Efficient Minimal Perfect Hash Functions**. Berlin, Heidelberg: Springer, 2007. 139–150 p. ISBN 978-3-540-73951-7. Disponível em: <http://dx.doi.org/10.1007/978-3-540-73951-7_13>. Cited on page 55.

BOYTSOV, L. Indexing Methods for Approximate Dictionary Searching: Comparative Analysis. **Journal of Experimental Algorithmics**, ACM, v. 16, p. 1–91, 2011. ISSN 1084-6654. Disponível em: <<https://doi.org/10.1145/1963190.1963191>>. Cited 7 times on pages 21, 31, 36, 41, 46, 48, and 64.

BOYTSOV, I. Super-Linear Indices for Approximate Dictionary Searching. In: **International Conference on Similarity Search and Applications (SISAP 2012)**. [s.n.], 2012. p. 162–176. Disponível em: <https://doi.org/10.1007/978-3-642-32153-5_12>. Cited 7 times on pages 20, 36, 37, 40, 55, 62, and 104.

BRILL, E.; MOORE, R. C. An Improved Error Model for Noisy Channel Spelling Correction. In: **38th Annual Meeting on Association for Computational Linguistics (ACL00)**. Stroudsburg, PA, USA: Association for Computational Linguistics, 2000. (ACL '00), p. 286–293. Disponível em: <<https://doi.org/10.3115/1075218.1075255>>. Cited on page 34.

CHEGRANE, I.; BELAZZOUGUI, D. Simple, compact and robust approximate string dictionary. **CoRR**, abs/1312.4678, 2013. Disponível em: <<http://dblp.uni-trier.de/db/journals/corr/corr1312.html#ChegraneB13>>. Cited 3 times on pages 37, 62, and 104.

CHURCH, K.; GALE, W. Probability Scoring for Spelling Correction. **Statistics and Computing**, v. 1, n. 2, p. 93–103, 1991. Disponível em: <[brokenurl#http://publication.wilsonwong.me/load.php?id=233281639](http://publication.wilsonwong.me/load.php?id=233281639)>. Cited on page 34.

CRISTO, M.; HANADA, R.; CARVALHO, A.; LORES, F. A.; PIMENTEL, M. d. G. C. Fast Word Recognition for Noise Channel-based Models in Scenarios with Noise Specific Domain Knowledge. In: **Proceedings of the 2017 ACM on Conference on Information and Knowledge Management**. New York, NY, USA: ACM, 2017. (CIKM '17), p. 607–616. ISBN 978-1-4503-4918-5. Disponível em: <<https://doi.org/10.1145/3132847.3133028>>. Cited 6 times on pages 26, 63, 65, 66, 67, and 68.

DAMERAU, F. J. A Technique for Computer Detection and Correction of Spelling Errors. **Communications of the ACM**, v. 7, n. 3, p. 171–176, 1964. ISSN 0001-0782. Disponível em: <<https://doi.org/10.1145/363958.363994>>. Cited 2 times on pages 32 and 33.

DEOROWICZ, S.; CIURA, M. Correcting spelling errors by modelling their causes. **International journal of applied mathematics and computer science**, University of Zielona Gora Press, v. 15, n. 2, p. 275–285, 2005. Disponível em: <<http://pldml.icm.edu.pl/pldml/element/bwmeta1.element.bwnjournal-article-amcv15i2p275bwm>>. Cited on page 63.

DIAZ-TULA, A.; MORIMOTO, C. H. Augkey: Increasing foveal throughput in eye typing with augmented keys. In: **Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems**. New York, NY, USA: ACM, 2016. (CHI '16), p. 3533–3544. ISBN 978-1-4503-3362-7. Disponível em: <<http://doi.acm.org/10.1145/2858036.2858517>>. Cited on page 38.

DOMINGUES, M. A.; SANTOS, T. E.; HANADA, R.; CUNHA, B. C.; REZENDE, S. O.; PIMENTEL, M. d. G. C. A platform for the recommendation of points of interest in brazilian cities: Architecture and case study. In: **Proceedings of the 21st Brazilian Symposium on Multimedia and the Web**. New York, NY, USA: ACM, 2015. (WebMedia '15), p. 229–236. ISBN 978-1-4503-3959-9. Disponível em: <http://doi.acm.org/10.1145/2820426.2820448>. Cited on page 26.

GARBE, W. **Improved Edit-distance based Spelling Correction**. 2012. Disponível em: <http://blog.faroo.com/2012/06/07/improved-edit-distance-based-spelling-correction/>. Cited 3 times on pages 37, 40, and 47.

GOLLAPUDI, S.; PANIGRAHY, R. A Dictionary for Approximate String Search and Longest Prefix Search. In: **15th ACM international conference on Information and knowledge management**. [s.n.], 2006. p. 768–775. ISBN 1-59593-433-2. Disponível em: <https://doi.org/10.1145/1183614.1183723>. Cited on page 29.

GREENLAND, S.; SENN, S.; ROTHMAN, K.; CARLIN, J.; POOLE, C.; GOODMAN, S.; ALTMAN, D. Statistical Tests, P Values, Confidence Intervals, and Power: a Guide to Misinterpretations. Springer Netherlands, v. 31, n. 4, p. 337–350, 2016. Cited on page 79.

HANADA, R.; CRISTO, M.; PIMENTEL, M. d. G. C. How do metrics of link analysis correlate to quality, relevance and popularity in wikipedia? In: **Proceedings of the 19th Brazilian Symposium on Multimedia and the Web**. New York, NY, USA: ACM, 2013. (WebMedia '13), p. 105–112. ISBN 978-1-4503-2559-2. Disponível em: <http://doi.acm.org/10.1145/2526188.2526198>. Cited on page 26.

HANADA, R.; PIMENTEL, M. d. G. C.; CRISTO, M.; LORES, F. A. Effective Spelling Correction for Eye-based Typing Using Domain-specific Information About Error Distribution. In: **25th ACM International on Conference on Information and Knowledge Management**. New York, NY, USA: ACM, 2016. (CIKM '16), p. 1723–1732. ISBN 978-1-4503-4073-1. Disponível em: <https://doi.org/10.1145/2983323.2983838>. Cited 11 times on pages 21, 23, 26, 35, 45, 48, 49, 50, 51, 52, and 53.

HIMORO, M. Y.; HANADA, R.; CRISTO, M.; PIMENTEL, M. d. G. C. An investigation of the relationship between the amount of extra-textual data and the quality of wikipedia articles. In: **Proceedings of the 19th Brazilian Symposium on Multimedia and the Web**. New York, NY, USA: ACM, 2013. (WebMedia '13), p. 333–336. ISBN 978-1-4503-2559-2. Disponível em: <http://doi.acm.org/10.1145/2526188.2526218>. Cited on page 26.

HUCKAUF, A.; URBINA, M. H. Gazing with peyes: Towards a universal input for various applications. In: **Proceedings of the 2008 Symposium on Eye Tracking Research & Applications**. New York, NY, USA: ACM, 2008. (ETRA '08), p. 51–54. ISBN 978-1-59593-982-1. Disponível em: <http://doi.acm.org/10.1145/1344471.1344483>. Cited on page 38.

ISLAM, A.; INKPEN, D. Real-word Spelling Correction Using Google Web 1Tn-gram Data Set. In: **18th ACM conference on Information and knowledge management**. [s.n.], 2009. p. 1689–1692. ISBN 978-1-60558-512-3. Disponível em: <https://doi.org/10.1145/1645953.1646205>. Cited on page 64.

ISOKOSKI, P. Text Input Methods for Eye Trackers Using Off-screen Targets. In: **2000 symposium on Eye tracking research & applications**. New York, NY, USA: ACM, 2000. (ETRA '00), p. 15–21. ISBN 1-58113-280-8. Disponível em: <https://doi.org/10.1145/355017.355020>. Cited 2 times on pages 19 and 37.

JURAFSKY, D.; MARTIN, J. H. Spelling Correction and the Noisy Channel. The Spelling Correction Task. In: **Speech and Language Processing**, 2nd Ed. Prentice-Hall, 2009. ISBN 0131873210. Disponível em: <<https://web.stanford.edu/~jurafsky/slp3/5.pdf>>. Cited 5 times on pages 20, 21, 29, 35, and 70.

KERNIGHAN, M. D.; CHURCH, K. W.; GALE, W. A. A Spelling Correction Program Based on a Noisy Channel Model. In: **13th conference on Computational linguistics - Volume 2**. [s.n.], 1990. p. 205–210. Disponível em: <<https://doi.org/10.3115/997939.997975>>. Cited 3 times on pages 20, 32, and 69.

KUMAR, M.; WINOGRAD, T.; PAEPCKE, A. Gaze-enhanced Scrolling Techniques. In: **CHI '07 Extended Abstracts on Human Factors in Computing Systems**. New York, NY, USA: ACM, 2007. (CHI EA '07), p. 2531–2536. ISBN 978-1-59593-642-4. Disponível em: <<https://doi.org/10.1145/1240866.1241036>>. Cited 2 times on pages 39 and 75.

KURAUCHI, A.; FENG, W.; JOSHI, A.; MORIMOTO, C.; BETKE, M. EyeSwipe: Dwell-free Text Entry Using Gaze Paths. In: **Conference on Human Factors in Computing Systems (CHI 2016)**. New York, NY, USA: ACM, 2016. (CHI '16), p. 1952–1956. ISBN 978-1-4503-3362-7. Disponível em: <<https://doi.org/10.1145/2858036.2858335>>. Cited 2 times on pages 20 and 39.

LEVENSHTEIN, V. Binary Codes Capable of Correcting Deletions, Insertions and Reversals (in russian). **Dokl. Akad. Nauk SSSR**, v. 163, p. 845–848, 1965. Disponível em: <<http://www.mathnet.ru/links/89ce49beef6d9601639ff587538796c2/dan31411.pdf>>. Cited on page 32.

LEVENSHTEIN, V. I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals (translated version). **Soviet Physics Doklady**, v. 10, p. 707–710, 1966. Disponível em: <<https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf>>. Cited 2 times on pages 32 and 33.

LUONG, T.; PHAM, H.; MANNING, C. D. Effective approaches to attention-based neural machine translation. In: MÀRQUEZ, L.; CALLISON-BURCH, C.; SU, J.; PIGHIN, D.; MARTON, Y. (Ed.). **EMNLP**. The Association for Computational Linguistics, 2015. p. 1412–1421. ISBN 978-1-941643-32-7. Disponível em: <<http://dblp.uni-trier.de/db/conf/emnlp/emnlp2015.html#LuongPM15>>. Cited on page 104.

MACKENZIE, I. S.; SOUKOREFF, R. W. Phrase Sets for Evaluating Text Entry Techniques. In: **CHI '03 Extended Abstracts on Human Factors in Computing Systems**. New York, NY, USA: ACM, 2003. (CHI EA '03), p. 754–755. ISBN 1-58113-637-4. Cited on page 84.

MAJARANTA, P.; AHOLA, U.-K.; SPAKOV, O. Fast Gaze Typing with an Adjustable Dwell Time. In: **SIGCHI Conference on Human Factors in Computing Systems (CHI 2009)**. New York, NY, USA: ACM, 2009. (CHI '09), p. 357–360. ISBN 978-1-60558-246-7. Disponível em: <<https://doi.org/10.1145/1518701.1518758>>. Cited 2 times on pages 19 and 38.

MAYS, E.; DAMERAU, F. J.; MERCER, R. L. Context Based Spelling Correction. **Information Processing & Management**, Elsevier, v. 27, n. 5, p. 517–522, 1991. ISSN 0306-4573. Disponível em: <[https://doi.org/10.1016/0306-4573\(91\)90066-U](https://doi.org/10.1016/0306-4573(91)90066-U)>. Cited 3 times on pages 29, 32, and 69.

MIHOV, S.; SCHULZ, K. U. Fast Approximate Search in Large Dictionaries. **Computational Linguistics**, MIT Press, v. 30, n. 4, p. 451–477, 2004. ISSN 0891-2017. Disponível em: <<https://doi.org/10.1162/0891201042544938>>. Cited on page 36.

MOR, M.; FRAENKEL, A. S. A Hash Code Method for Detecting and Correcting Spelling Errors. **Communications of the ACM**, v. 25, n. 12, p. 935–938, 1982. ISSN 0001-0782. Disponível em: <<https://doi.org/10.1145/358728.358752>>. Cited 2 times on pages 20 and 36.

MOTT, M. E.; WILLIAMS, S.; WOBBROCK, J. O.; MORRIS, M. R. Improving Dwell-Based Gaze Typing with Dynamic, Cascading Dwell Times. In: **Conference on Human Factors in Computing Systems**. New York, NY, USA: ACM, 2017. (CHI '17), p. 2558–2570. ISBN 978-1-4503-4655-9. Disponível em: <<https://doi.org/10.1145/3025453.3025517>>. Cited on page 39.

NORVIG, P. **How to Write a Spelling Corrector**. 2007. Disponível em: <<http://norvig.com/spell-correct.html>>. Cited on page 23.

NORVIG, p. Natural Language Corpus Data. In: SEGARAN, T.; HAMMERBACHER, J. (Ed.). **Beautiful Data: The Stories Behind Elegant Data Solutions**. O'Reilly Media, 2009. ISBN 0596157118. Disponível em: <<http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0596157118>>. Cited 14 times on pages 20, 21, 22, 34, 40, 43, 44, 47, 48, 61, 64, 71, 76, and 78.

PEDROSA, D.; PIMENTEL, M. D. G.; WRIGHT, A.; TRUONG, K. N. Filtered typing: Design Challenges and User Performance of Dwell-Free Eye Typing. **ACM Transactions on Accessible Computing**, ACM, New York, NY, USA, v. 6, n. 1, p. 1–37, mar. 2015. ISSN 1936-7228. Disponível em: <<https://doi.org/10.1145/2724728>>. Cited 13 times on pages 20, 21, 23, 30, 39, 40, 47, 83, 84, 85, 87, 88, and 100.

PETROU, M.; PETROU, C. Reducing low frequency interference. In: **Image Processing: The Fundamentals**. Ed. 2. [S.l.]: John Wiley & Sons, 2010. ISBN 978-0-470-74586-1. Cited 2 times on pages 21 and 48.

PI, J.; SHI, B. E. Probabilistic adjustment of dwell time for eye typing. In: **10th International Conference on Human System Interactions (HSI)**. [s.n.], 2017. p. 251–257. Disponível em: <<https://doi.org/10.1109/HSI.2017.8005041>>. Cited on page 39.

POWELL, D. R.; ALLISON, L.; DIX, T. I.; DOWE, D. L. Alignment of Low Information Sequences. In: **4th Australasian Theory Symposium (CATS '98)**. [S.l.: s.n.], 1998. p. 215–230. Cited on page 33.

RAMOS, G. M. A.; HANADA, R.; PIMENTEL, M. da G. C.; TEIXEIRA, C. A. C. A Word-prediction Eye-typing Approach for Brazilian Portuguese Entries using Geometric Movements. In: **35th ACM International Conference on the Design of Communication (SIGDOC 2017)**. [s.n.], 2017. p. 1–6. Disponível em: <<https://doi.org/10.1145/3121113.3121234>>. Cited 2 times on pages 26 and 38.

SARCAR, S.; PANWAR, P.; CHAKRABORTY, T. EyeK: An Efficient Dwell-free Eye Gaze-based Text Entry System. In: **11th Asia Pacific Conference on Computer Human Interaction**. New York, NY, USA: ACM, 2013. (APCHI '13), p. 215–220. ISBN 978-1-4503-2253-9. Disponível em: <<https://doi.org/10.1145/2525194.2525288>>. Cited 4 times on pages 19, 20, 38, and 88.

TOPAL, C.; BENLIGIRAY, B.; AKINLAR, C. SliceType: Fast Gaze Typing with a Merging Keyboard. **CoRR**, abs/1706.02499, 2017. Disponível em: <<http://arxiv.org/abs/1706.02499>>. Cited on page 38.

TULA, A. D.; CAMPOS, F. M. S. de; MORIMOTO, C. H. Dynamic context switching for gaze based interaction. In: **Proceedings of the Symposium on Eye Tracking Research and Applications**. New York, NY, USA: ACM, 2012. (ETRA '12), p. 353–356. ISBN 978-1-4503-1221-9. Disponível em: <<http://doi.acm.org/10.1145/2168556.2168635>>. Cited on page 37.

TULA, A. D.; KURAUCHI, A. T.; MORIMOTO, C. H. Facilitating gaze interaction using the gap and overlap effects. In: **CHI '13 Extended Abstracts on Human Factors in Computing Systems**. New York, NY, USA: ACM, 2013. (CHI EA '13), p. 91–96. ISBN 978-1-4503-1952-2. Disponível em: <<http://doi.acm.org/10.1145/2468356.2468374>>. Cited on page 38.

VOORHEES, E. M. The TREC-8 Question Answering Track Report. In: VOORHEES, E. M.; HARMAN, D. K. (Ed.). **Text REtrieval Conference (TREC) 1999 Spoken Document Retrieval Track**. (NIST), 1999. Disponível em: <http://trec.nist.gov/pubs/trec8/papers/qa_report.pdf>. Cited 3 times on pages 39, 64, and 79.

WAGNER, R.; LOWRANCE, R. An Extension of the String-to-String Correction Problem. **J. ACM**, v. 22, n. 2, p. 177–183, 1975. ISSN 0004-5411. Disponível em: <<https://doi.org/10.1145/321879.321880>>. Cited on page 33.

WARD, D. J.; MACKAY, D. J. C. Fast Hands-free Writing by Gaze Direction. **Nature**, v. 418, n. 6900, p. 838, 2002. Disponível em: <<http://www.inference.phy.cam.ac.uk/mackay/abstracts/eyeshortpaper.html>>. Cited 2 times on pages 37 and 38.

WITTEN, I. H.; FRANK, E. **Data Mining: Practical Machine Learning Tools and Techniques**. Ed. 3. Morgan Kaufmann, 2011. Disponível em: <<http://amazon.com/o/ASIN/1558605525/>>. Cited 2 times on pages 48 and 63.

WOBBROCK, J. O.; RUBINSTEIN, J.; SAWYER, M. W.; DUCHOWSKI, A. T. Longitudinal Evaluation of Discrete Consecutive Gaze Gestures for Text Entry. In: **Symposium on Eye Tracking Research & Applications**. New York, NY, USA: ACM, 2008. (ETRA '08), p. 11–18. ISBN 978-1-59593-982-1. Disponível em: <<https://doi.org/10.1145/1344471.1344475>>. Cited 2 times on pages 19 and 38.