AN IMPROVED LEVENSHTEIN ALGORITHM FOR SPELLING CORRECTION WORD CANDIDATE LIST GENERATION

HANAN NAJM ABDULKHUDHUR

COLLEGE OF ARTS AND SCIENCE

UNIVERSITI UTARA MALAYSIA

2016

Permission to Use

In presenting this dissertation in fulfillment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the Universiti Library may make it freely available for inspection. I further agree that permission for the copying of this dissertation in any manner, in whole or in part, for the scholarly purpose may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying or publication or use of this dissertation or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my dissertation.

Requests for permission to copy or to make other use of materials in this dissertation, in whole or in part should be addressed to:

Dean of Awang Had Salleh Graduate School of Arts and Sciences
UUM College of Arts and Sciences
Universiti Utara Malaysia
06010 UUM Sintok

Abstrak

Senarai calon terhasil dalam pembetulan ejaan adalah satu proses untuk mencari kata-kata dari leksikon yang hampir sama dengan perkataan yang tidak tepat. Algoritma paling banyak digunakan untuk menjana senarai calon untuk kata-kata yang tidak tepat adalah berdasarkan jarak Levenshtein. Walau bagaimanapun, algoritma ini mengambil masa yang terlalu lama apabila terdapat bilangan besar kesilapan ejaan. Sebabnya ialah bahawa pengiraan algoritma Levenshtein termasuk operasi yang menghasilkan jajaran dan pengisian sel-sel jajaran dengan membandingkan huruf-huruf perkataan yang tidak betul dengan huruf-huruf perkataan dari leksikon. Oleh kerana kebanyakan leksikon mengandungi berjuta-juta perkataan, maka operasi ini akan diulang berjuta-juta kali bagi setiap perkataan tidak tepat untuk menjana senarai calonnya. Kajian ini men ambahbaikkan algoritma Levenshtein dengan merekabentuk teknik operasi yang telah dimasukkan dalam algoritma ini. Teknik operasi yang dicadangkan meningkatkan algoritma Levenshtein dari segi masa pemprosesan perlaksanaannya tanpa menjejaskan ketepatannya. Ia mengurangkan langkah operasi yang diperlukan untuk mengukur nilai sel-sel dalam baris dan lajur pertama, baris dan lajur kedua serta baris dan lajur ketiga dalam jajaran Levenshtein. Algoritma Levenshtein yang telah tingkatkan telah dibandingkan dengan algoritma asal. Hasil kajian menunjukkan bahawa prestasi algoritma yang dicadangkan melebihi prestasi algoritma Levenshtein asal dari segi masa pemprosesan, iaitu sebanyak 36.45% manakala ketepatan kedua-dua algoritma adalah masih sama.

Kata Kunci: Algoritma Levenshtein, Masa pemprosesan, Penghasilan senarai calon,

Pmbetulan Kesilapan.

Abstract

Candidates' list generation in spelling correction is a process of finding words from a lexicon that should be close to the incorrect word. The most widely used algorithm for generating candidates' list for incorrect words is based on Levenshtein distance. However, this algorithm takes too much time when there is a large number of spelling errors. The reason is that calculating Levenshtein algorithm includes operations that create an array and fill the cells of this array by comparing the characters of an incorrect word with the characters of a word from a lexicon. Since most lexicons contain millions of words, then these operations will be repeated millions of times for each incorrect word to generate its candidates list. This dissertation improved Levenshtein algorithm by designing an operational technique that has been included in this algorithm. The proposed operational technique enhances Levenshtein algorithm in terms of the processing time of its executing without affecting its accuracy. It reduces the operations required to measure cells' values in the first row, first column, second row, second column, third row, and third column in Levenshtein array. The improved Levenshtein algorithm was evaluated against the original algorithm. Experimental results show that the proposed algorithm outperforms Levenshtein algorithm in terms of the processing time by 36.45% while the accuracy of both algorithms is still the same.

Keywords: Levenshtein Algorithm, Processing time, Candidates' list generation, Errors correction.

Acknowledgement

Each part of this dissertation is guided, inspired, and supported by many people. Firstly, I would like to thank all the members of my family especially my husband and my parents for their unconditional support. My goal would not be achieved without them. The most important support and guidance were from my research supervisors Dr. Shahrul Azmi Mohd Yusof and Prof. Madya Dr. Yuhanis Yusof. Thank you very much for your great help and support. It is an honor for me to do a research under your supervisions. I would like to thank all the academic and technical staff in School of Computing of Utara Universiti Malaysia for their help in the study process and providing all the excellent facilities. Finally, I would like to thank all my friends for their support.

Table of Contents

Permission to Use i

Abstrak ii

Abstract iii

Acknowledgement iv

Table of Contents v

List of Tables viii

List of Figures ix

List of Appendices x

List of Abbreviations xi

CHAPTER ONE INTRODUCTION 1

1.1 Background 1

1.2 Problem Statement 4

1.3 Research Questions 6

1.4 Research Objectives 6

1.5 Significance of the Dissertation 6

1.6 Scope of the Dissertation 7

1.7 Organization of the Dissertation 8

CHAPTER TWO LITERATURE REVIEW 9

2.1 Introduction 9

2.2 Spelling Correction Stages 9

2.2.1 Error Detection Stage 9

2.2.2 Candidates List Generation Stage 10

2.2.3 Error Correction Stage 11

2.3 Techniques of Candidates List Generation Based on Edit Distance 11

2.3.1 Levenshtein Distance 12

2.3.2 Hamming Distance 16

2.3.3 Damerau-Levenshtein Distance 17

2.3.4 Longest Common Subsequence Distance 17

2.5.5 Threshold Levenshtein Distance 18

2.5.6 Levenshtein Automata Distance 18

2.4 Other Techniques of Candidates List Generation 20

2.4.1 N-gram Distance 20

2.4.2 Bag Distance 21

2.4.3 N-gram language Model 21

2.4.4 Topic Model 22

2.5 Evaluation Measurement 23

2.6 Summary 25

CHAPTER THREE RESEARCH METHODOLOGY 26

3.1 Introduction 26

3.2 Research Phases 26

3.3 Theoretical Study 27

3.4 Design Phase 28

3.5 Development Phase 30

3.6 Evaluation 30

3.6.1 Experimental Design 30

3.6.1.1 How to Measure Processing Time 31

3.6.1.2 How to Measure Accuracy 33

3.6.1.3 Statistical Test 35

3.6.2 Testing Dataset 36

3.7 Summary 38

CHAPTER FOUR PROPOSED DIFFERENTIATION TECHNIQUE 39

4.1 Introduction 39

4.2 ILA-OT Steps 39

4.2.1 First Step 40

4.2.2 Second Step 41

4.2.3 Third Step 45

4.3 Comparison between operations of LA and ILA-OT 49

4.4 ILA-OT Algorithm 55

4.5 Summary 60

CHAPTER FIVE EXPERIMENTAL RESULTS AND DISCUSSION 62

5.1 Introduction 62

5.2 Processing Time 62

5.3 Accuracy 66

5.4 Statistical Test 67

5.5 Results Discussion 68

5.6 Summary 69

CHAPTER SIX CONCLUSION AND FUTURE WORK 70

6.1 Introduction 70

6.2 Research Summary... 70

6.3 Research Contributions 71

6.4 Recommendation for Future Work 73

6.5 Summary 74

REFERENCES 75

APPENDICES 80

List of Tables

Table 3.1 Main variables resulted from t-test. 36

Table 3.2 Characteristics of the testing dataset. 37

Table 4.1 Theoretical comparison between operations of LA and ILA-OT. 52

Table 4.2 Practical comparison between LA and ILA-OT. 54

Table 5.1 A sample of the processing time values of both algorithms 62

Table 5.2 Processing times of both algorithms LA and ILA-OT. 64

Table 5.3 A sample of the distances of both algorithms for several words. 66

Table 5.4 T-test results between LA and ILA-OT. 67

List of Figures

Figure 1.1. The scope of this research. 7

Figure 2.1. Levenshtein distance example 13

Figure 2.2. Bigram distance example.. 20

Figure 3.1. Research phases... 27

Figure 4.1. Levenshtein array before and after applying the first step. 40

Figure 4.2. The second step examples for the first row. 42

Figure 4.3. The second step examples for the first column. 44

Figure 4.4. Third step examples for the second row.. 46

Figure 4.5. The third step examples for the second column. 48

Figure 4.6. Cells affected by three steps proposed for this dissertation..................... 54

Figure 5.1. Line graph for the processing time values listed in Table 5.2................. 65

List of Appendices

Appendix A Sample of Output Results of Word Length 3. 80

Appendix B Sample of Output Results of Word Length 4. 81

Appendix C Sample of Output Results of Word Length 5. 82

Appendix D Sample of Output Results of Word Length 6. 83

Appendix E Sample of Output Results of Word Length 7. 84

Appendix F Sample of Output Results of Word Length 8.. 85

Appendix G Sample of Output Results of Word Length 9. 86

Appendix H Sample of Output Results of Word Length 10. 87

Appendix I Sample of Output Results of Word Length 11.. 88

Appendix J Sample of Output Results of Word Length 12. 89

List of Abbreviations

LA	Levenshtein algorithm
HD	Hamming distance
DD	Damerau distance
LCS	Longest common subsequence
OT	Operational technique
ILA-OT	Improved Levenshtein algorithm by using the proposed OT
MVFRFC	Measure values of the first row and first column
MVSRSC	Measure values of the second row and second column
PT	Processing time
PD	Percentage decrease

CHAPTER ONE
INTRODUCTION

1.1 Background

Spelling correction is the process of detecting and repairing spelling errors in a text. Research in spelling correction is not new; it started in the mid of 1960, and many algorithms for spelling correction have been suggested since then (Mahdi, 2012). Spelling correction can be either manual or automatic. The first type allows intervention of humans in the correction process. The second type, a system will decide the correction to replace an incorrect word by choosing the best candidate word without human's intervention (Bassil & Alwani, 2012b).

Most methods of automatic spelling correction have three functions: error detection, generation of candidates, and error correction (Naseem & Hussain, 2007). The first function is to find incorrect words in the output text. The second function is to generate candidate words from a lexicon for each of the incorrect words. Candidate list generation is a process of finding words from a lexicon that should be close to the incorrect word. For example, the candidates' list generated from a lexicon for the incorrect word "*czp*" are "*cup*", "*cap*", and "*cop*". The last function is to correct all incorrect words by selecting the best candidate to replace with each incorrect word.

The process of generating candidates list can be achieved by using a specific algorithm. An algorithm is a set of operations that will be performed on some data to solve a specific problem. In general, algorithms can be classified according to their optimal solution into two categories: exact and approximate. In execution, exact algorithms will reach an optimal solution while approximation algorithms can be used to find a solution close to the optimal solution. Approximation algorithms are used in cases where finding the optimal solution is impractical, or when exact algorithms take too much time in execution (Sheng, Tao, & Li, 2012).

Candidates' list generation requires too much time when there is a large number of spelling errors (Al-Bakry & Al-Rikaby, 2015; Al-Masoudi & Al-Obeidi, 2015; Al-Zaydi & Salam, 2015). For example, the OCR error rate can reach up to 60% for noisy images (Lund, 2014; Ma & Agam, 2013). Therefore, a normal book with 100,000 words will require 60,000 corrections, which indicate that any improvement in processing time of generating candidates' list for each error can increase the performance of these systems.

Levenshtein algorithm is one of the exact algorithms, and many researchers used it to generate a list of candidates for incorrect words from language resources (Adhitama, Kim, & Na, 2014; Al-Zaydi & Salam, 2015; Daðason, 2012). In general, Levenshtein distance is designed for measuring the difference between any two strings. In other words, it counts the minimum number of operations required to transform one string to another (Levenshtein, 1966; Lounis, Guermeche, Eddine, Saoudi, & Benaicha, 2014).

The operations of Levenshtein distance are performed on a single symbol, and they consist from insertion, deletion, and substitution. The term "symbol" represents the smallest meaningful unit in a writing system, such as character, number, comma, punctuation, signs, etc. Each operation of a single symbol is considered as a single edit (Navarro, 2001). For example, "*Michccl*" is a non-word in English, because it does not exist in this language. It requires two substitutions to become "*Michael*" that is considered a correct word in English (Grzebala, 2016). Levenshtein algorithm is used in many applications. However, this dissertation focuses on improving Levenshtein edits distance when it used in candidate list generation.

To generate candidates' list by using Levenshtein algorithm, it requires creating a two-dimensional array and fills each cell in this array by comparing characters of an incorrect word with characters of a word from a lexicon (Phillips, 2015). Furthermore, Levenshtein algorithm needs to calculate million times for each incorrect word because most lexicons contain millions of words (Bassil & Alwani, 2012b). Each cell value in Levenshtein array needs almost eight operations: three comparing, three adding and two assigning (Al-Bakry & Al-Rikaby, 2015).

Operations of each cell in Levenshtein array include two parts. The first part is used to measure a variable value named cost. This will be done by one comparing operation, to check if there is a match between a character in an incorrect word and a character of a word in a lexicon and one assigning operation to set the value of the variable cost. The value of cost is 0 if both characters are match and 1 otherwise. The second part includes six operations: three adding, two comparing, and one assigning. The adding operations are one to increase the value of left cell by one and another adding operation to increase the value of an upper cell by one, and the last adding is to increase the value of a diagonal previous cell by the value of cost. The comparing operations are two to find minimum value among three cells: diagonal previous cell, left's cell, and upper's cell. The assigning operation includes storing the resulted value in a related cell of Levenshtein array (Adhitama et al., 2014; Haldar & Mukhopadhyay, 2011; Lounis et al., 2014). The eight operations required to measure each cell value multiplied by the number of cells in Levenshtein array will be repeated for all words listed in a lexicon to generate candidates' list for a single incorrect word.

1.2 Problem Statement

Several solutions have been proposed to solve the processing time problem of generating candidates list for a large number of errors, but most of them are based on using approximate distances such as n-gram distance (Bassil & Alwani, 2012b). The approximate distance may sometimes produce incorrect candidates list. They are used when a correction process allows a tolerance of some error to correct a large number of errors quickly. Other solutions are based on using exact algorithms. However, each algorithm suffers different limitation, and selecting the appropriate algorithm depends on where it will be used (Gomaa & Fahmy, 2013; Vargas, 2008). For example, longest common subsequence (LCS) algorithm did not result in a very relevant match for the incorrect word in most cases as more than one word have the same distance by the LCS (Gupta, Bhatt, & Mittal, 2016)

Levenshtein algorithm is widely used in finding the nearest correct words from a lexicon to the incorrect word due to its accuracy (Adhitama et al., 2014; Al-Zaydi & Salam, 2015; Daðason, 2012). However, using Levenshtein algorithm requires too much time when there is a large number of spelling errors (Al-Bakry & Al-Rikaby, 2015; Al-Masoudi & Al-Obeidi, 2015; Al-Zaydi & Salam, 2015). For example, the processing time to generate candidates list for incorrect words "*affort*" and "*usre*" from a lexicon, which contains 500k words, using Levenshtein algorithm are 19.1 and 17.2 seconds respectively (Al-Bakry & Al-Rikaby, 2015). This means the processing time to generate candidates list for 100 incorrect words is almost (17.2 * 100=1720 seconds=28.6 minutes).

The processing time of Levenshtein algorithm is large because calculating it requires operations to create an array and to fill each cell in this array by comparing characters of an incorrect word with characters of a word from a lexicon (Al-Bakry & Al-Rikaby, 2015; Lounis et al., 2014; Phillips, 2015). Several works have been done on Levenshtein algorithm to be reliable for specific purposes. However, some of these works make Levenshtein algorithm approximate in order to be executed quickly, such as the work proposed by (Andoni & Krauthgamer, 2012; Andoni & Onak, 2012). Unfortunately, no experiments are performed to measure how much the accuracy is dropped or how much the processing time is increased. Navarro, Grabowski, Mäkinen, and Deorowicz (2005) improved Levenshtein algorithm to be faster for music information retrieval by ignoring deletion and insertion operations from the calculation. The author only considers transposition operation because of characteristics of music pieces. The improved Levenshtein algorithm becomes faster and reliable only for measuring the difference between two pieces of music. However, it is unreliable for spelling correction due to the deletion and insertion occurred in incorrect words. Other works make Levenshtein algorithm able to perform additional tasks in addition to its original task, such as the work proposed by (Pal & Rajasekaran, 2015) but it becomes slower than the original due to the additional code added to it.

To conclude, it is a need to design a technique that can help Levenshtein algorithm in reducing the operations needed to measure values of each cell in the Levenshtein array without affecting its accuracy.

1.3 Research Questions

In order to reduce computational cost of the Levenshtein algorithm for generating candidate words for spelling errors, the following questions need to be addressed:

i. How to shorten the operations needed to measure values of the cells in Levenshtein algorithm array without affecting its accuracy?

ii. How to develop and evaluate the improved Levenshtein algorithm?

1.4 Research Objectives

The goal of this dissertation is to develop an improved Levenshtein algorithm that speeds up the process of generating candidate words for spelling correction. The specific objectives are:

i. To design an operational technique to shorten the operations of the cells in the Levenshtein array without affecting its accuracy.

ii. To develop and evaluate the improved Levenshtein algorithm that includes the proposed operational technique.

1.5 Significant of the Dissertation

As a theoretical effect of this dissertation, an improved Levenshtein algorithm has been designed. The improved algorithm reduces the processing time required to generate candidate words for spelling errors. The improved algorithm includes the proposed operational technique that decreases operations required to measure each cell value in Levenshtein array. Furthermore, the proposed operational technique is new, and it is not adopted from other topics. Therefore, it may be considered as a guideline for researchers to suggest or improve this technique in similar work in other topics.

As a practical effect of this dissertation, many applications such as DNA searching, sequences' alignment, word-processing programs, speech processing systems, and optical character recognition systems use Levenshtein edit distance algorithm in their components. Therefore, any improvement in this algorithm can contribute to these applications. The aim is to reduce the processing time of the execution of these applications.

1.6 Scope of the Dissertation

The scope of this dissertation is shown in Figure 1.1.

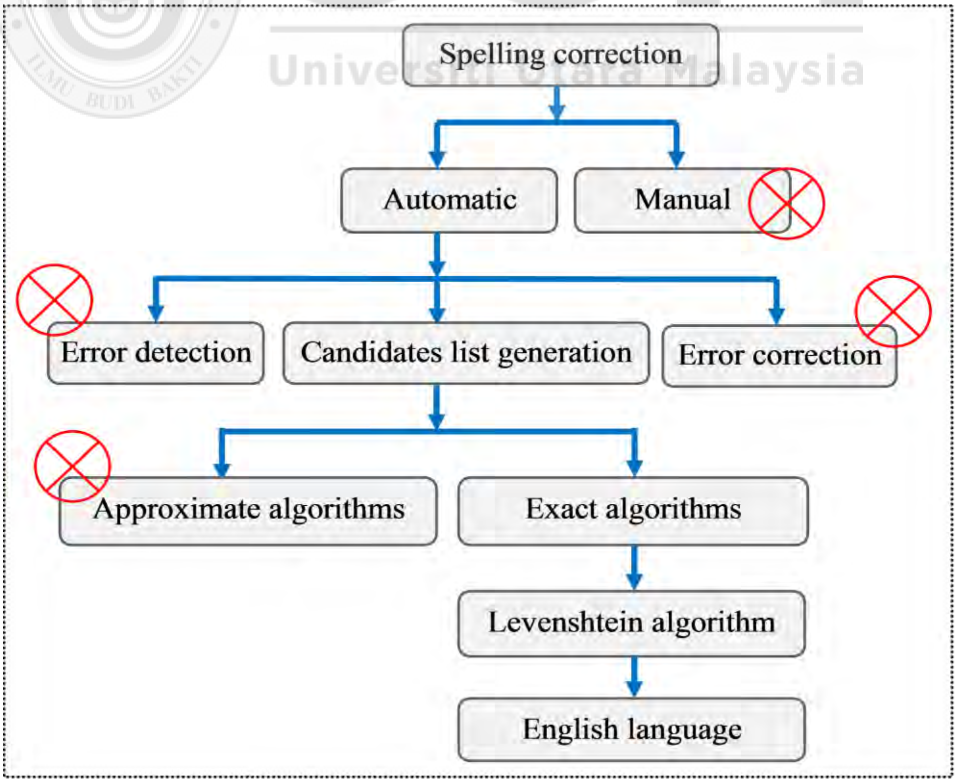


Figure 1.1. The scope of this dissertation

Figure 1.1 shows that this dissertation has been performed under the category of automatic spelling correction. Furthermore, it only deals with generating of candidates list from this category. In addition to that, Levenshtein algorithm, which is an exact algorithm, is selected to be improved in terms of its processing time. The proposed Levenshtein algorithm has been tested only for the English language.

1.7 Organization of the Dissertation

The dissertation report contains six chapters. Chapter one includes the introduction, problem statement, and necessary information to understand the concepts that can be used in the later chapters. Chapter two discussed the literature review with a description of the different aspects relating to the research area. Chapter three focuses on a methodology that was used in the research. Chapter four presents the design of the improved Levenshtein algorithm while chapter five shows the experimental results of the proposed algorithm. Finally, chapter six presents the conclusion and future work of this dissertation.

CHAPTER TWO
LITERATURE REVIEW

2.1 Introduction

This chapter presents the review of previous studies that have been done to speed up the processing time of generating candidates' list in spelling correction. Many studies try to solve this problem using different techniques and algorithms. This chapter reviews some of these studies and points out how they evolved and developed. In addition, their limits and gaps have been highlighted. A literature review is based primarily on recently published work of articles in journals, conferences, reports, books, and thesis.

This chapter organized into six main sections. Spelling correction stages are given in Section 2.2. The candidates list generation techniques that based on edit distance are discussed in Section 2.3 while other techniques for candidates' list generation are explained in Section 2.4. The evaluation measurements used by other researchers are presented in Section 2.5. At the end of this chapter, a summary is shown in Section 2.6.

2.2 Spelling Correction Stages

This section discusses stages of spelling correction, which are error detection, candidates' list generation, and error correction.

2.2.1 Error Detection Stage

The error detection is usually classified into two categories: isolated word-based detection, and context-based detection (Formiga Fanals & Rodríguez Fonollosa, 2012). In the first category, the process of detection is based only on an incorrect word itself. An incorrect word is processed in isolation without giving any attention to the context. This category can detect only non-word errors. Non-word errors occur when the word produced from a human writing or from a program execution, does not exist in the language resource, such as the word "*foed*". In the second category, the process of detection is based on incorrect word and the context of the sentence. It is used to detect real word error and non-word errors. The real word error occurs when the word produced from a human writing or from a program execution, exists in the language resource, but it does not match with the source text, such as the word "*too*" in the sentence "*I want too eat*" (Habeeb, Yusof, & Ahmad, 2014).

2.2.2 Candidates List Generation Stage

Candidates' list generation are usually classified into two categories: character-based generation, and word-based generation (Vargas, 2008). In the first category, the process of generating candidates list is based on the characters of the words. This category is slow in computing, but it does not require predefined rules or large training corpus. In the second category, the process of generating candidates list is based only on the words without giving any attention to their characters. This category is fast in computing, but it requires predefined rules or large training corpus (Bassil & Alwani, 2012b).

Most techniques used in generating candidates list are based on string distance algorithms (Polyanovsky, Roytberg, & Tumanyan, 2011). As mentioned previously, the term "*distance"* means the different between two strings. Some candidates' list techniques are fast, but they are approximate, i.e, they may not find the best solution, such as heuristic techniques or probabilistic techniques (Cooper & Cooper, 1981).

Other techniques are accurate, but they are slow, such as those based on dynamic programming approach (Al-Bakry & Al-Rikaby, 2015). Approximate techniques are proposed for online search engines, where they do not ensure that the research results represent an exact match (Polyanovsky et al., 2011). All main techniques used in candidates' list generation are discussed in detail in Section 2.3 and Section 2.4.

2.2.3 Error Correction Stage

The input to this stage is a candidates list. It uses a procedure to select the best suggestion from the candidates list to replace with an incorrect word. This process will be repeated until all incorrect words are corrected. When the auto correction is implemented, the incorrect word is automatically replaced by the first ranking word in an ordered candidates' list (Naseem & Hussain, 2007).

The choice of the appropriate technique for ranking is very important. The reason is that it may replace the incorrect word with another word that can be found in the language resource, but it is unsuitable for the sentence, resulting the desired goal of correction is to be unachieved. Error correction stage usually contains two techniques, one to correct non-word error and the other to correct real word error (Bassil & Alwani, 2012a; Daðason, 2012).

2.3 Techniques of Candidates List Generation Based on Edit Distance

As mentioned in chapter one, the term "edit distance" is used for calculating the difference between two strings. The operations are performed on a single symbol, and they consist from insertion, deletion, transposition, or substitution (Lu, Du, Hadjieleftheriou, & Ooi, 2014). The description of four edit distance operations is illustrated below:

- **Insertion of a single symbol**: if a string is equal to the "*xy"*, then inserting the symbol z in this string will produce "*xzy*".
- **Deletion of a single symbol:** if a string is equal to the "*xy*", then deleting the symbol y in this string will produce "*x*".
- **Substitution of a single symbol**: if a string is equal to the "*xy*", then substituting the symbol y by z in this string will produce "xz".
- **Transposition of a single symbol**: if a string is equal to the "*xy*", then transposition these symbols in this string will produce "yx".

Main algorithms that based on edit distance to generate candidates' list are explained in next sections.

2.3.1 Levenshtein Distance

Levenshtein distance was proposed by the Russian scientist Vladimir Levenshtein (Levenshtein, 1966). The definition of Levenshtein distance between two tokens or two sequences is "the minimum number of insertions, deletions or substitutions needed to change one token or one sequence into the other". Levenshtein distance can deal with three edit operations: insertions, deletions, and substitutions (Attia, Pecina, Samih, Shaalan, & van Genabith, 2012; Phillips, 2015). Figure 2.1 shows an example of how to calculate Levenshtein edit distance between two tokens *"CLARKE" and "CLERK"*.

		C	L	A	R	K	E
	0	1	2	3	4	5	6
C	1	**0**	1	2	3	4	5
L	2	1	**0**	1	2	3	4
E	3	2	1	1	2	3	**3**
R	4	3	2	2	**1**	2	3
K	5	4	3	3	2	**1**	2

Figure 2.1. Levenshtein distance example, (Source: Phillips, 2015)

Figure 2.1 illustrates that it is a need for two edit operations to convert the token "*CLARKE*" to the token "*CLERK*" or vice versa as shown in the last cell in the array of this figure. Figure 2.1 also shows that an array is initialized to measure edit distance between the tokens. The array can be filled from the upper left to the lower right corner. Each jump horizontally or vertically corresponds to an insert or a delete, respectively. The cost is normally set to 1 for each of the operations. The diagonal jump can cost either one if the two characters in the row and column do not match or 0 if they do. Each cell always minimizes the cost locally. This way the number in the lower right corner is the Levenshtein distance between both tokens. The pseudo-code used to implement Levenshtein edit distance, programmatically, is presented in Algorithm 2.1 (Lounis et al., 2014).

Algorithm 2.1: Levenshtein algorithm, Source: (Lounis et al., 2014)

```
1   integer DistanceOfLevenshtein (string chaine1, string chaine2)
    // i and j iterate over chaine1 and chaine2
2   Integer i, j, cost, lengthChaine1, lengthChaine2
3   lengthChaine1:= length(Chaine1)
4   lengthChaine2 := length(Chaine2)
    // d is a table of lengthChaine1 + 1 rows and lengthChaine2 + 1 columns
5   Integer d[0..lengthChaine1, 0..lengthChaine2]
6   for i from 0 to lengthChaine1
7           d[i, 0] := i
8   for j from 0 to lengthrChaine2
9           d[0, j] := j
10  for i from 1to lengthChaine1
11          for j from 1to lengthChaine2
12              if chaine1[i - 1] = chaine2[j - 1] then cost := 0
13              else cost := 1
14              d[i, j] := minimum(
                                 d[i-1, j ] + 1,          // deletion
                                 d[i, j-1] + 1,           // insertion
                                 d[i-1, j-1] + cost       // substitution
                                     )
15  return d[lengthrChaine1, lengthChaine2]
```

The Levenshtein algorithm receives two strings as inputs: "*chaine1*" and "*chaine2*". A two-dimensional array named "*d*" is created as shown in step 5. This array has been used to measure edit distance between *"chaine1" and "chaine2"*. To refer to any cell in *"d"* array, two variables are used named "i" and "j". The variable "i" refers to the position of a row in "*d*" array while variable "j" refers to the position of a column in the "*d*" array. After that, Algorithm 2.1 show that cells values of the first row and first column will be measured by using counter while other cells values in other rows and columns will be measured by using two loops. Each cell in

Levenshtein array includes measuring the value of a variable called cost as shown in steps 12 and 13. This will be done by checking if there is a match between a character in first string and a character of the second string. The value of cost is 0 if both characters are match and 1 otherwise. The next step is to find minimum value among three cells: diagonal previous cell left the cell, and upper cell (Lounis et al., 2014). After that, the edit distance between *"chaine1" and "chaine2"* will be in the last cell in the *"d"* array, which is *d[lengthrChaine1, lengthChaine2]*.

Several researchers have been improved Levenshtein algorithm to be reliable for specific purposes. For example, Pal and Rajasekaran (2015) improved Levenshtein algorithm to find motif in a set of biological strings. The motif is a substring that appears in a set of input biological strings. The characters in motif can be not neighbored. For example, the string "GT***G" is a motif. The symbol "*" can be referred to any character in sequence. However, other characters should appear in all input strings with the same context. This improvement makes Levenshtein algorithm suitable for finding motif in biological strings. However, it becomes slower than the original due to the additional code added to it. Unfortunately, experimental results have been done without comparing with original Levenshtein algorithm. Therefore, it is difficult to measure the difference in terms of processing time of both algorithms.

Navarro et al. (2005) improved Levenshtein algorithm to be faster for music information retrieval by ignoring deleting and inserting operations from the calculation. The author only considers transposition operation because of characteristics of music pieces. These characteristics do not allow deleting and inserting operations in music pieces. Therefore, the author ignores them from the calculation. The improved Levenshtein algorithm becomes faster and reliable only for measuring the difference between two pieces of music. However, it is unreliable for spelling correction due to the deletion and insertion occurred in incorrect words.

Other improvements in Levenshtein algorithm includes making it approximate in order to be executed quickly, such as the works done in (Andoni & Krauthgamer, 2012; Andoni & Onak, 2012; Burkhardt & Kärkkäinen, 2002; Huldén, 2009; Mihov & Schulz, 2004). As mentioned previously, the approximate algorithms may sometimes produce incorrect candidates list while the goal of this dissertation is to improve Levenshtein algorithm in terms of processing time without affecting its accuracy.

2.3.2 Hamming Distance (HD)

Hamming distance was proposed by the scientist Richard Hamming. It uses to measure the difference between two tokens. It is applied only if the two tokens have an equal number of symbols. Otherwise, it cannot be calculated if the two tokens have a different number of symbols (Singh et al., 2015). The reason is that it measures only substitution operation in the same position in both tokens, and it ignores other operations of edit distance. In other words, it calculates the minimum number of substitutions needed to change one token into the other (Ruoro, 2009; Singh et al., 2015).

Hamming distance is faster than Levenshtein distance because it only deals with substitution operation from four edit distance operations. However, Levenshtein distance is better than Hamming distance because it can deal with two tokens having different lengths.

2.3.3 Damerau-Levenshtein Distance

This distance is similar to the standard Levenshtein distance. However, the difference between them is that Damerau-Levenshtein distance can deal with four primitive editing distance instead of three in the standard Levenshtein distance (Li, Zhang, Zhu, & Zhou, 2006). The definition of Damerau distance between two tokens or two sequences is “the minimum number of insertions, deletions, substitutions or transposition needed to change one token or one sequence into the other”. In other words, this distance added transposition edit operation to the standard Levenshtein distance (Bard, 2007). Damerau-Levenshtein distance is slower than Levenshtein distance due to the additional code required to measure transposition edit operation (Al-Bakry & Al-Rikaby, 2015).

2.3.4 Longest Common Subsequence Distance

Longest common subsequence (LCS) distance can only deal with insertion and deletion operations from four primitive edit distance operations. It is used to find the longest common subsequence between two tokens (Bergroth, Hakonen, & Raita, 2000). A subsequence that appears in two tokens is considered as a common subsequence of these two tokens. In other words, an LCS is a common subsequence having a maximum length of shared symbols.

Levenshtein distance is better than LCS distance because LCS can produce in some cases wrong results. For example, the distances between the string “culess” and three other strings (“colourless”, “cluelessness”, and “cloudless”) using LCS are 7, 7, and 7 respectively (Gupta et al., 2016) while they are 3, 5, and 2 respectively by using Levenshtein distance. Therefore, it can select the string “cloudless” as the nearest word to the string “culess” if Levenshtein distance is used. In contrast, it is difficult to select the nearest word to the string “culess” if LCS is used due to the similar results.

2.3.5 Threshold Levenshtein Distance

The speed of standard Levenshtein distance was improved by the scientist Esko Ukkonen. He makes Levenshtein distance takes two strings and a variable called threshold. Next, the difference between them is measured with a condition that the difference should be less or equal to the threshold. In other words, it does not need to complete any extra comparison between two strings if the difference exceeds the value of the threshold. This will speed up generating candidates list for any incorrect word (Mihov & Schulz, 2004; Ukkonen, 1985). However, this distance will be restricted by a threshold (Mihov & Schulz, 2004).

2.3.6 Levenshtein Automata Distance

Levenshtein automata distance is depending on finite state theory. It is proposed to avoid calculating of Levenshtein edit distance so that process of generating of candidates list becoming fast (Hassan, Noeman, & Hassan, 2008). Using finite state theory in generating candidates list is performed by using two steps. The first step will replace each character in misspelled word with all characters of the specific language. Therefore, a temporary list of candidates will be produced. However, the resulted candidates list may contain incorrect words, and this will lead to the step two.

In step two, a procedure will be performed to filter temporary candidates list with all words in a lexicon. Any word in temporary candidates' list does not exist in a lexicon will be excluded in order to produce final candidates list (Huldén, 2009; Mihov & Schulz, 2004). In other words, this technique takes the time to compute Levenshtein automata for the wrong word to generate a temporary set of candidates (V). Then it takes the time to parallel or filter V with all words in the lexicon (W). Filter means comparing each word in V with each word in W.

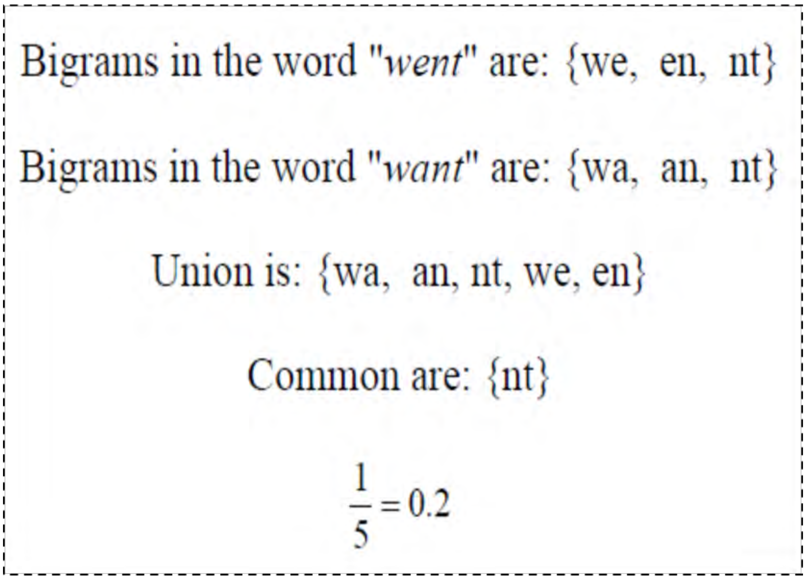
For example, by assuming alphabet has only 30 symbols and there is a single incorrect word containing 10 characters, then using finite state theory will generate temporary candidates list of 639 tokens if distance is one, about 400,000 tokens if distance is two, and about 260,000,000 tokens if distance is three (Mihov & Schulz, 2004). Therefore, generating temporary candidates list using finite state theory is only tested with misspelled word having one or two errors. This is because filtering 639 tokens or 400,000 tokens with all words in a lexicon are different than filtering 260,000,000 tokens with the same number of words in a lexicon. The English language contains 26 small characters and 26 capital characters. Therefore, generating temporary candidates list using Levenshtein automata distance for incorrect words having more than two errors takes long processing time.

The advantage of the Levenshtein automata distance is: it avoids calculating Levenshtein edit distance in order to generate of candidates list quickly. However, the disadvantage is: it is only tested with misspelled word having one or two errors (Hassan et al., 2008; Huldén, 2009; Mihov & Schulz, 2004; Vargas, 2008). Therefore, it needs to test with tokens having three errors or more because each extra error will increase temporary candidates list that needs to be filtered with all words in a lexicon to produce final candidates list. Furthermore, this distance is not reliable for some language, such as Han Chinese because it contains 70,000 Unicode Han characters (Ahmed, 2015).

2.4 Other Techniques of Candidates List Generation

2.4.1 N-gram Distance

N-gram distance is an approximate distance using to measure the similarity between two sequences of strings. The value of the first character in term "N-gram" can be 1, 2, 3, 4,..., n. The term itself represents a sequence of N neighboring symbols in a token. N-gram is called unigram when N=1, bigram when N=2, trigram when N=3, and so on. The accuracy of similarity increases when the value of N is high and vice versa. The similarity is measured by dividing the number of shared N-grams of the two sequences by the total number of N-grams in same two sequences. Figure 2.2 shows an example of how can calculate the similarity between two tokens "went" and "want" using bigrams distance (Bassil & Alwani, 2012a; Naseem, 2004).



Bigrams in the word "*went*" are: {we, en, nt}

Bigrams in the word "*want*" are: {wa, an, nt}

Union is: {wa, an, nt, we, en}

Common are: {nt}

$$\frac{1}{5} = 0.2$$

Figure 2.2. Bigram distance example, (Source: Bassil & Alwani, 2012b)

The accuracy of Levenshtein distance is better than N-gram distance. This is because N-gram distance does not show good accuracy on short strings (Naseem, 2004). For example, trigram distance will not give any similarity if two strings have length three, and there are one or two different symbols between them. For instance, despite the presence of two letters are similar between tokens “crp” and “cup”, trigram will not show any similarity between them.

2.4.2 Bag Distance

Bag distance is an approximation to the edit distance. It does not give accuracy as Levenshtein distance. However, it is faster than Levenshtein distance (Vargas, 2008). The term bag represents a token or a sequence. For example, by assuming two strings named S_1 and S_2 and S_1=“abcd” and S_2=“abcxz”, then bag distance is the characters “xz”. The reason is that the bag distance technique first removed common characters from S1 and S2, and this will result in two new sequences named S3=“d” and S4=“xz”. After that, the bag distance will become equal to the sequence S4 because S4 has a number of characters greater than S3.

The advantage of the bag distance is that it is faster than Levenshtein distance. However, it is less performance than edit-distance family because it is similar to the N-gram distance does not take into consideration the importance of the positions of characters in both tokens (Vargas, 2008).

2.4.3 N-gram language Model

N-grams language model is a statistical model. It used to provide a probability for a sequence of words (Jurafsky & Martin, 2009). A probability depended on the frequency of words or frequency of sentences in a large corpus. The language model can be used to detect and correct incorrect words or only to generate candidates list (Habeeb et al., 2014). Potential incorrect word error will happen if the probability of sentence provided by the language model is almost zero. A correction of this error is based on the high probability of other sentences. The most important point in this approach is that large corpus is needed in order to build an accurate language model.

A major problem facing N-gram language model is that it depends on the finite training corpus. Therefore, some tokens will be missed from the corpus. They are called unknown tokens. If any N-gram is missing, then the language model will give a probability of zero to it, and no candidates' list will be generated. (Jurafsky & Martin, 2009).

2.4.4 Topic Model

The idea of a topic model is based on creating multiple dictionaries from a corpus or from the web. Each one belongs to specific topics, such as medical, sports, etc (Wick, Ross, & Learned-Miller, 2007). In spelling correction, every document needed to be checked for errors is classified according to its topic. After that, candidates' list is generated for each incorrect word from a dictionary that has the same topic.

The advantage of the topic model is: it is fast in generating candidates list. However, the disadvantage is: it has the same problem of an N-gram language model, which is it needs a very large corpus for each topic. Furthermore, it is difficult to identify a topic type for missing words in the corpus. For example, if the document needed to be check belongs to the sports topic. Therefore, all candidates list will be generated from the dictionary of the same topic. This will lead to the problem of missing some words from sports dictionary because they are not appearing in a web or in a corpus when creating a sports dictionary. However, they could exist in other dictionaries of other topics (Wick et al., 2007).

2.5 Evaluation Measurement

This section discusses the evaluation measurements used by other researchers in this kind of domain, especially for Levenshtein algorithm. In automatic spelling correction, evaluation measurements used by most researchers are based mainly on two metrics: accuracy and processing time of errors correction. Some researchers used only one of these metrics, such as accuracy (Mainsah, Morton, Collins, Sellers, & Throckmorton, 2015; Siklósi, Novák, & Prószéky, 2016) or processing time (Al-Bakry & Al-Rikaby, 2015) and others used both metrics in their work (Popescu & Vo, 2014).

As mentioned previously, Levenshtein algorithm can produce candidates' list for incorrect word accurately because it is not an approximate algorithm. Therefore, the accuracy metric in automatic spelling correction is not based on generating candidates' list performed by Levenshtein algorithm but on algorithms combined with it to detect and correct errors in a text. In other words, if there is an algorithm to detect the errors, Levenshtein algorithm to generate candidates' list, and algorithm to correct the error by selecting the best candidate and replaces it with the incorrect word, then the accuracy metric will be based on detection algorithm and correction algorithm. For previous reason, it is better to evaluate Levenshtein algorithm separately.

Since Levenshtein algorithm can measure the distance between any two strings accurately. Therefore, there is a little work on this algorithm in terms of accuracy. In the following sentences and paragraphs, this dissertation has been described some works done by other researchers that related to the Levenshtein algorithm. Al-Bakry and Al-Rikaby (2015) compared between three algorithms, which are Levenshtein algorithm, Damerau-Levenshtein algorithm and enhanced Damerau-Levenshtein algorithm using only processing time as metric. The experimental results show that processing time of Levenshtein algorithm is almost the same to the processing time of enhanced Damerau-Levenshtein algorithm. Since the output of Levenshtein algorithm is different than outputs of other algorithms used in the evaluation, then, it is difficult to measure the accuracy among them.

Maarif, Akmeliawati, Htike, and Gunawan (2014) also compared between five algorithms, which are Levenshtein algorithm, N-gram algorithm, Soundex algorithm, Mahalanobis algorithm and Jaro-Wrinkler algorithm using only processing time as metric. The experimental results show that the processing time of N-gram algorithm is the best among other algorithms. However, as mentioned previously, N-gram algorithm is approximate algorithm while Levenshtein algorithm is an exact algorithm. The accuracy metric is not considered in the evaluation process due to the different output of each algorithm.

Rieck and Wressnegger (2016) developed a tool called “Harry” for measuring the similarity between strings. The tool “Harry” can measure the similarity using one of many algorithms including Levenshtein algorithm. In addition to measure the similarity between strings, it can calculate run-time of each algorithm, which means processing time, with a different number of available CPU cores. This tool also cannot measure the accuracy due to the different output of each algorithm.

Pradhan, Gyanchandani, and Wadhvani (2015) presented a review on text similarity techniques used in information retrieval and its applications. Authors claimed that this review represents the most popular text similarity techniques. The review compares between nine algorithms including Levenshtein algorithm. Unfortunately, the comparison does not use the metrics of processing time and accuracy but it was performed based on the work of each algorithm.

To conclude, Levenshtein algorithm cannot be compared with other algorithms in terms of accuracy due to the different output (distance) for each algorithm. The evaluation metrics used by this dissertation has been described in Section 3.6.1.

2.6 Summary

This chapter began with explaining spelling correction stages that were explored by researchers in this field. The Levenshtein algorithm example, its pseudo-code, and improvement by other researchers were presented and discussed. Other techniques that were particularly used for candidates list generation was presented and discussed in detail. This is to give a glance of the problem and nature of solutions that are presented in this dissertation. As can be seen in this chapter the topic of candidates list generation has been discussed by various researchers for different purposes. This gives a clear picture of the importance of this topic. The last section of this chapter is a discussion on evaluation measurements used by other researchers in this kind of domain, especially for Levenshtein algorithm.

CHAPTER THREE
RESEARCH METHODOLOGY

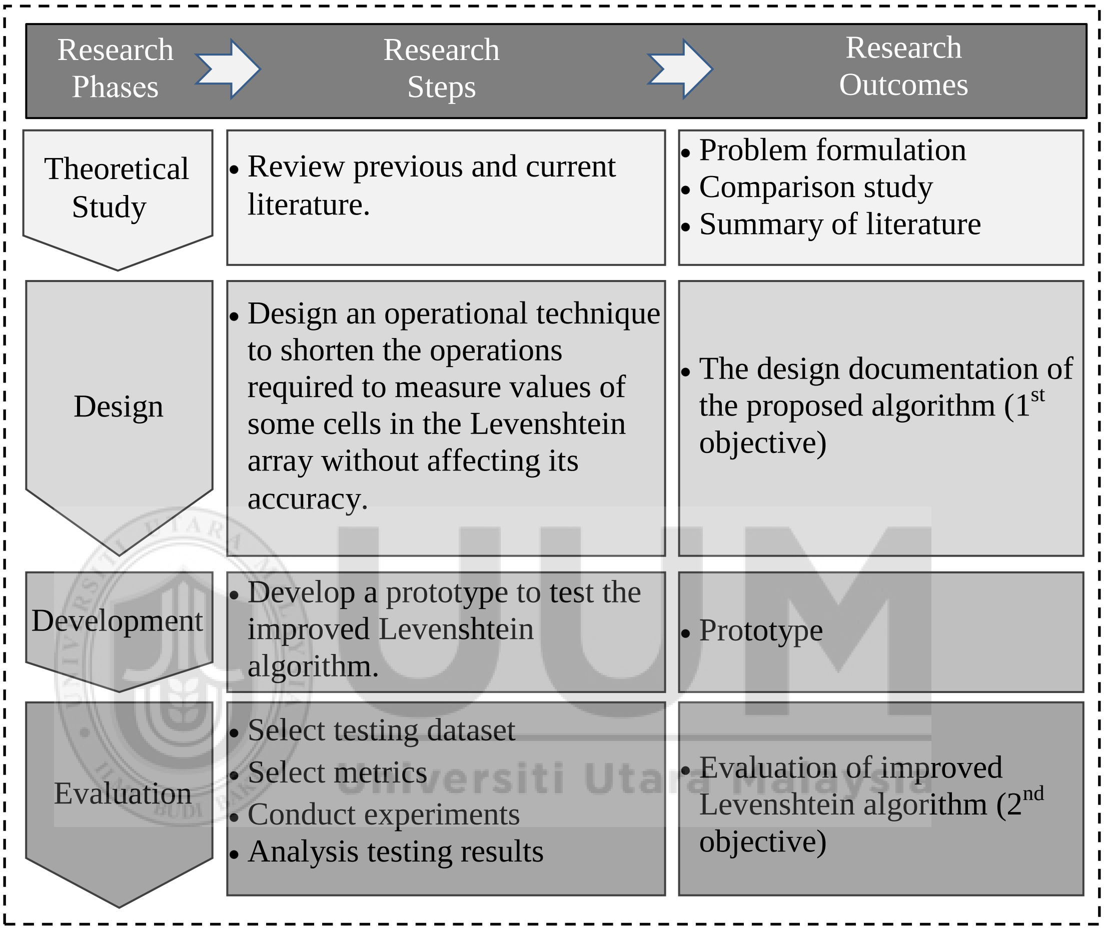
3.1 Introduction

Research methodology is used to explain the steps needed to achieve objectives of the specific study. Existing techniques that use in generating candidates list for spelling correction are developed based on experimental approach (Navarro, 2001). Therefore, this dissertation followed the same approach in developing improved Levenshtein algorithm. Chapter three includes Section 3.2 that explains the research phases. This is followed by Section 3.3 that presents the theoretical study of this dissertation. Section 3.4 presents the design of the improved Levenshtein algorithm while the development of this algorithm is explained in Section 3.5. The last phase undertakes evaluation as discussed in Section 3.6. Finally, a summary of chapter three is presented in Section 3.7.

3.2 Research Phases

Figure 3.1 shows that this research consists of four phases. They are the theoretical study, design, development, and evaluation. These phases have been adopted from (Alobaedy, 2015). Figure 3.1 also shows the main activities involved in each phase required for achieving research objectives and the outcome of each activity. Overall, objective one has been achieved by finding a solution that can shorten the operations required to measure values of some cells in the Levenshtein array without affecting its accuracy. Then, a technique has been designed based on this solution. The detail of this technique is explained in Section 3.3. On the other hand, objective two has been achieved by developing and evaluating the improved Levenshtein algorithm.

The evaluation process includes designing and implementing a prototype for testing the proposed algorithm, experimental design, testing dataset, data analysis and reporting final results. Next sections describe the details of each research phase.



Research Phases	Research Steps	Research Outcomes
Theoretical Study	• Review previous and current literature.	• Problem formulation • Comparison study • Summary of literature
Design	• Design an operational technique to shorten the operations required to measure values of some cells in the Levenshtein array without affecting its accuracy.	• The design documentation of the proposed algorithm (1st objective)
Development	• Develop a prototype to test the improved Levenshtein algorithm.	• Prototype
Evaluation	• Select testing dataset • Select metrics • Conduct experiments • Analysis testing results	• Evaluation of improved Levenshtein algorithm (2nd objective)

Figure 3.1. Research phases

3.3 Theoretical Study

The initial step in this dissertation was the theoretical study. In this step, two directions are used in analyzing the research problem. The first direction focuses on the state of the art in generating candidates list for spelling correction. This information is extracted from literature obtained from different types of publications; conference proceedings, technical reports, books, thesis, and journals, with a focus on recent publications. The purpose of this information is to identify the problem statement, research questions, research objectives, research significance, and research scope.

The second direction focuses on studying in depth the techniques used in generating candidates list for spelling correction, especially about Levenshtein distance. This direction includes several steps: identifying weakness and strength of these techniques, selecting the best among them, and make a critical review, which means comparing them and summarize the results. The information resulted from this direction is important because if these techniques and their limitations are understood, then there is a chance to improve one of them.

3.4 Design Phase

This dissertation refers to the improved Levenshtein algorithm as ILA-OT. The term ILA-OT means improved Levenshtein algorithm by using the proposed operational technique. As mentioned previously, this technique reduces the operations required to measure cells' values in Levenshtein array. The ILA-OT algorithm is based on the idea that it can remove first row and first column from LA array. Furthermore, it can predict cells' values in a second row, second column, third row, and third column in LA array without needed a traditional way. This is because there is a context to measure cells values in these rows and columns until finding a shared character in them. After finding a shared character, the context will be changed, therefore, it can measure cells values in these rows and columns based on the new context. Lastly, the context will only be changed once, and it will not change if there is an additional shared character in them.

Since an average word length for The English language is 5 characters (Lund, 2014), then Levenshtein array will be 36 cells (6 rows * 6 columns). Thereby, operations of 3 rows from six and operations of 3 columns from six will be affected by the ILA-OT idea. In other words, operations of 27 cells will be decreased from 36 in Levenshtein array. This certainly leads to reduce the processing time of Levenshtein algorithm execution. Three steps have been derived from the ILA-OT idea.

The first step removes first row and first column from Levenshtein array. This is because values of the first row and first column are used in measuring values of second row and second column. Therefore, measuring values of the second row and the second column directly will lead ignoring them from Levenshtein array. Measuring values of the second row and the second column has been done by the second step. This dissertation finds a context in measuring values of second row and second column. This context requires fewer operations to measure values of second row and second column. This context has been formulated as the equation for step two.

The third step also finds a context in measuring values of third row and third column. However, the context of the second step is different from the context of the third step. The context of the third step needs values of second row and the second column to measure values of third row and third column while the context of the second step does not require values from any row or column. This context of the third step has also been formulated as the equation for the third step. Expected processing time of Levenshtein algorithm has been decreased if the proposed three steps of this dissertation are performed. This is because filling of Levenshtein array becomes requiring fewer operations.

Lastly, since the design of the proposed steps and their contributions required pseudo-code of proposed algorithm, examples, and descriptions to explain them. A separate chapter (i.e. chapter four) is presented.

3.5 Development Phase

This stage developed a prototype for measuring the processing time and accuracy for the original and improved Levenshtein algorithm. This development includes choosing a programming language, selecting a testing dataset, and an operating environment. The prototype has been developed using the followings:

- Visual studio Express 2012 technology, using VB.Net language. It can be downloaded free from Microsoft website.
- Microsoft windows seven 64 bits for an operating environment.
- Personal laptop HP has CPU of type Intel (TM) with 2400GHz while the RAM of the laptop is 8GB.

3.6 Evaluation

An evaluation process of improved Levenshtein algorithm includes experimental design, measurements, and testing dataset. Each part has been described in the following subsections.

3.6.1 Experimental Design

Experimental approach includes experimental design, measurement, and reporting. Experimental design defines the goals of tests and how they are achieved. A measurement focuses on what metrics are used in the evaluation process. Reporting focused on how can represent the results, for example, by using tables, graphs, etc (Rardin & Uzsoy, 2001).

Experiments of this dissertation have been conducted to achieve one goal. The goal is to reduce the processing time of the LA without affecting its accuracy. To achieve this goal, this dissertation compares the accuracy and processing time of the ILA-OT with accuracy and processing time of the LA. Hence, two metrics have been used in comparison: processing time and the accuracy.

In addition to both metrics used in the evaluation of ILA-OT, this dissertation conducted a statistical test to show if the improvement in the terms of the processing time of the ILA-OT is significant or not. A statistical test of the difference has been measured using t-test. This test is a statistical method of comparing two samples in terms of their values. It shows the mean of each group and the average difference between the groups. It also shows whether this difference is statistically significant or due to chance and other circumstances. Next sub-sections show how each metric is measured and how the statistical test is conducted.

3.6.1.1 How to Measure Processing Time

In each experiment, the time before and after executing each algorithm have been measured. Furthermore, the processing time has been measured using a built-in class named "*Stopwatch*" in visual basic.net programming language. This class receives two times and measures the difference (period) in time between them. In addition to that, it produces the time in minutes, seconds or milliseconds according to the user's desire. Algorithm 3.1 shows the steps used to measure processing time of both LA and ILA-OT.

Algorithm 3.1: Steps to Measure Processing Time

```
Define array called “PT_LA” that uses to store processing times of LA
Define array called “PT_ILA_OT” that uses to store processing times of ILA-OT
Read testing dataset file
Define array called “W” that uses to store words resulted from splitting dataset text file into words
Define sum and period as double variables
// the array “W” contains 10001 words and numbers of tests are 10000
For i from 0 to (W.length-2) do
    // measuring processing time of LA
    sum=0
    For j from 0 to 4 do                                    // repeat 5 times
        Record start time
        LA (W[i], W[i+1])                                   // Execute LA algorithm
        Record stop time
        period=the time between stop time and start time as double value
        sum=sum + period
    Next j
    PT_LA[i]= sum / 5                                       // take average value
    // measuring processing time of ILA-OT
    sum=0
    For j from 0 to 4 do                                    // repeat 5 times
        Record start time
        ILA_OT (W[i], W[i+1])                   // Execute ILA_OT algorithm
        Record stop time
        period=the time between stop time and start time as double value
        sum=sum + period
    Next j
    PT_ILA_OT[i]= sum / 5                                   // take average
Next i
Write   PT_LA array values to the excel file
Write  PT_ILA_OT array values to the excel file
End
```

Algorithm 3.1 shows that the processing time of each algorithm has been measured individually but for the same input words. It shows that a list of words was read from dataset file. Next, a word W(i) from this list and next word W(i+1) from the same list are sent firstly to the LA, and then to the ILA-OT for measuring processing time of each algorithm. The variable "i" represents the position of a word in the testing dataset while the term "W(i)" represents a word at position "i". Algorithm 3.1 also shows that the processing time of each input, which represents two different words, was measured five times as presented in steps 7 to 14 and the average are taken to represent the processing time as presented in step 15. The last steps in Algorithm 3.1 are to write the processing time values of both LA and ILA-OT to the excel file.

The percentage decrease (PD) in processing time has been calculated using Equation 3.1 (Lund, 2014).

$$\text{Percentage decrease in PT} = \frac{\text{PT(LA)} - \text{PT(ILA-OT)}}{\text{PT(LA)}} * 100 \qquad (3.1)$$

Where the term "PT(LA)" represents the average processing time of LA for all word lengths (10000 values) while the term "PT(ILA-OT)" represents the average processing time of ILA-OT for all word lengths (10000 values). Furthermore, the percentage decrease in processing time for each word length (1000 values) has been also calculated using same equation. The calculation of percentage decrease has been performed using Microsoft Excel 2010.

3.6.1.2 How to Measure Accuracy

Algorithm 3.2 shows the steps used to measure the accuracy between LA and ILA-OT.

Algorithm 3.2: Steps to Measure Accuracy

```
Define array called “distanceLA” that uses to store the outputs (distances) of LA
Define array called “distanceILA_OT” that uses to store the outputs (distances) of ILA-OT
Define a variable called “count_equal_distance” that uses to count how many both algorithms have the same output (distance)
Set count_equal_distance=0
Read  testing dataset file
Define array called “W” that uses to store words resulted from splitting dataset text file into words
// the array “W” contains 10001 words and numbers of tests are 10000
For i from 0 to (W.length-2) do
  distanceLA [i]=LA(W[i], W[i+1])                  // measure LA distance
  distanceILA_OT [i]= ILA_OT(W[i], W[i+1])   // measure ILA_OT distance
      If  distanceLA [i]= distanceILA_OT [i]
             count_equal_distance = count_equal_distance + 1
      End if
Next i
Write  distanceLA array values to the excel file
Write  distanceILA_OT array values to the excel file
Write  count_equal_distance value to the excel file
End
```

Algorithm 3.2 has some similar steps as Algorithm 3.1. It shows that a list of words was read from dataset file in the first step. Next, a word W(i) from this list and next word W(i+1) from the same list are sent firstly to the LA, and then to the ILA-OT for measuring the distance of each algorithm for these words. The variable “i” represents the position of a word in the testing dataset while the term “W(i)” represents a word at position “i”. Algorithm 3.2 also shows that if the values of outputs (distances) for both algorithms are equal, then a variable named “count_equal_distance” will be increased by one while if both algorithms have different distance, then this will consider an error. After that, the processes of measuring the distance of LA and measuring the distance of ILA-OT have been repeated for all words in this list. The last steps in Algorithm 3.2 are to write the distances values of both LA and ILA-OT and "count_equal_distance" value to the excel file.

The accuracy has been measured using Equation 3.2 (Batawi & Abulnaja, 2012) .

$$\text{Accuracy} = \frac{\text{Correct outputs}}{\text{Total outputs}} * 100 \tag{3.2}$$

Where the variable "Correct outputs" represents the number of equal distances between LA and ILA-OT algorithms. In other words, its value is equal to the value of variable "count_equal_distance" in Algorithm 3.2. The variable "Total output" represents the total number of comparisons between LA and ILA-OT, which is 10000 comparisons.

3.6.1.3 Statistical Test

As mentioned previously, a statistical test for the obtained results has been performed using t-test. This dissertation uses Microsoft Excel 2010 to analysis data, and to perform a t-test. Table 3.1 shows the main variables resulted from t-test with a description for each one.

Table 3.1

Main variables resulted from t-test

No.	**Variables**	**Description**
1	Mean	Mean of a set
2	Variance	Variance of the difference between the two means
3	Observations	Number of items in a set
4	t Stat	The t value that needs to be greater than t critical in order for the difference between the means to be significant
5	P(T<=t) two-tail	p-value: the probability that the difference between two means is real and not due to chance
6	t Critical two-tail	The value that needs to exceed by (t-Stat) in order for the difference between the means to be significant at the 5% level

Table 3.1 shows that the most important variables are p-value in row 5, and “t Critical” in row 6. P-value means the probability that the difference between two means is real and not due to chance. “t Critical” in row 6 is a value that needs to exceed by “t-Stat” in row 4. P-value should be less than 0.05, and “t Stat” value should be greater than t-critical in order for the difference between the means to be significant. Otherwise, it is not true and it is not real (Crumrine, Ritschel, & White, 2014; Musa, Opara, Shayib, & Oliver, 2016).

3.6.2 Testing Dataset

Most researchers use different sizes and types of the testing dataset in spelling correction (Navarro, 2001). However, Google recently has provided unigram files to be downloaded (Google, 2015). Therefore, this dissertation used these unigram files to test the improved Levenshtein algorithm. They have been used in many topics, such as spelling correction (Bassil & Alwani, 2012a; Islam & Inkpen, 2009) and machine translation (Federico & Cettolo, 2007). The words in Google unigram files are the most frequently used in the English language. Google unigram files are large and contain millions of words. Therefore, this dissertation followed the scenario proposed by (Hassan et al., 2008) in arranging the testing dataset. The characteristics of the testing dataset are shown in Table 3.2.

Table 3.2

Characteristics of the testing dataset

Word length	**Number of words in each length**
3	1000
4	1000
5	1000
6	1000
7	1000
8	1000
9	1000
10	1000
11	1000
12	1000
Total words in the testing dataset (from 3-to-12)	10000

Table 3.2 shows that this dissertation used single testing dataset containing 10000 words. Furthermore, the words in the testing dataset have 10 different word lengths, from 3 to 12, and each length contains 1000 words. The reason for the dataset

contains different word length is to analysis the impact of improved Levenshtein algorithm for each length of words.

3.7 Summary

This chapter provided a detailed description of the research methodology of this dissertation. This dissertation follows a methodology that is used in developing the most successful algorithms, which known as an experimental methodology. This kind of methodology can accept the feedback when some modifications are needed. The chapter began with a description of the research phases. The discussions on theoretical study, design phase, and development phase were presented. This is followed by explaining evaluation process. The explanation includes a discussion on evaluation measures and experimental settings. Statistical significance testing was suggested as well to ensure the results are statistically significant.

CHAPTER FOUR
IMPROVED LEVENSHTEIN ALGORITHM

4.1 Introduction

As was pointed out in previously, the improved Levenshtein algorithm is referred as ILA-OT. This chapter discussed the details of the ILA-OT algorithm. The chapter begins with introducing the steps of ILA-OT which is the core of the algorithm proposed in this dissertation. An example is provided for each step to show how ILA-OT measures cells values in its array. The comparison between LA and ILA-OT regarding the operations required to measure each cell value in their arrays is presented. The chapter continues with a discussion on the algorithms to implement ILA-OT. Finally, the chapter ends with a summary.

4.2 ILA-OT Steps

In this section, the details of the steps of the ILA-OT algorithm are explained. As mentioned previously, the ILA-OT algorithm is based on the idea that it can remove first row and first column from LA array. Furthermore, it can predict cells values in a second row, second column, third row, and third column in LA array without needed traditional way. This is because there is a context to measure cells values in these rows and columns until finding a shared character in them. After finding a shared character, the context will be changed, therefore, it can measure cells values in these rows and columns based on the new context. Lastly, the context will only be changed once, and it will not change if there is an additional shared character in them.

Based on the ILA-OT idea mentioned above, the proposed algorithm designs three steps. The first step deals with the first row and first column in LA array while the second step deals with the second row and second column in LA array. Lastly, the third step deals with a third row and third column in LA array. The result of combining these three steps is the ILA-OT algorithm. As compared to the LA, the proposed algorithm in this work eliminates first row and first column of the LA array. Furthermore, it measures values of the second row, second column, third row, and the third column with fewer operations. The three proposed steps are described in the following subsections.

4.2.1 First Step

Figure 4.1 shows Levenshtein array before and after applying the first step.

		b	a	c	k
	0	1	2	3	4
b	1	**0**	1	2	3
o	2	1	1	2	3
o	3	2	2	2	3
k	4	3	3	3	**2**

A: Before applying the first step

	b	a	c	k
b	**0**	1	2	3
o	1	1	2	3
o	2	2	2	3
k	3	3	3	**2**

B: After applying the first step

Figure 4.1. Levenshtein array before and after applying the first step

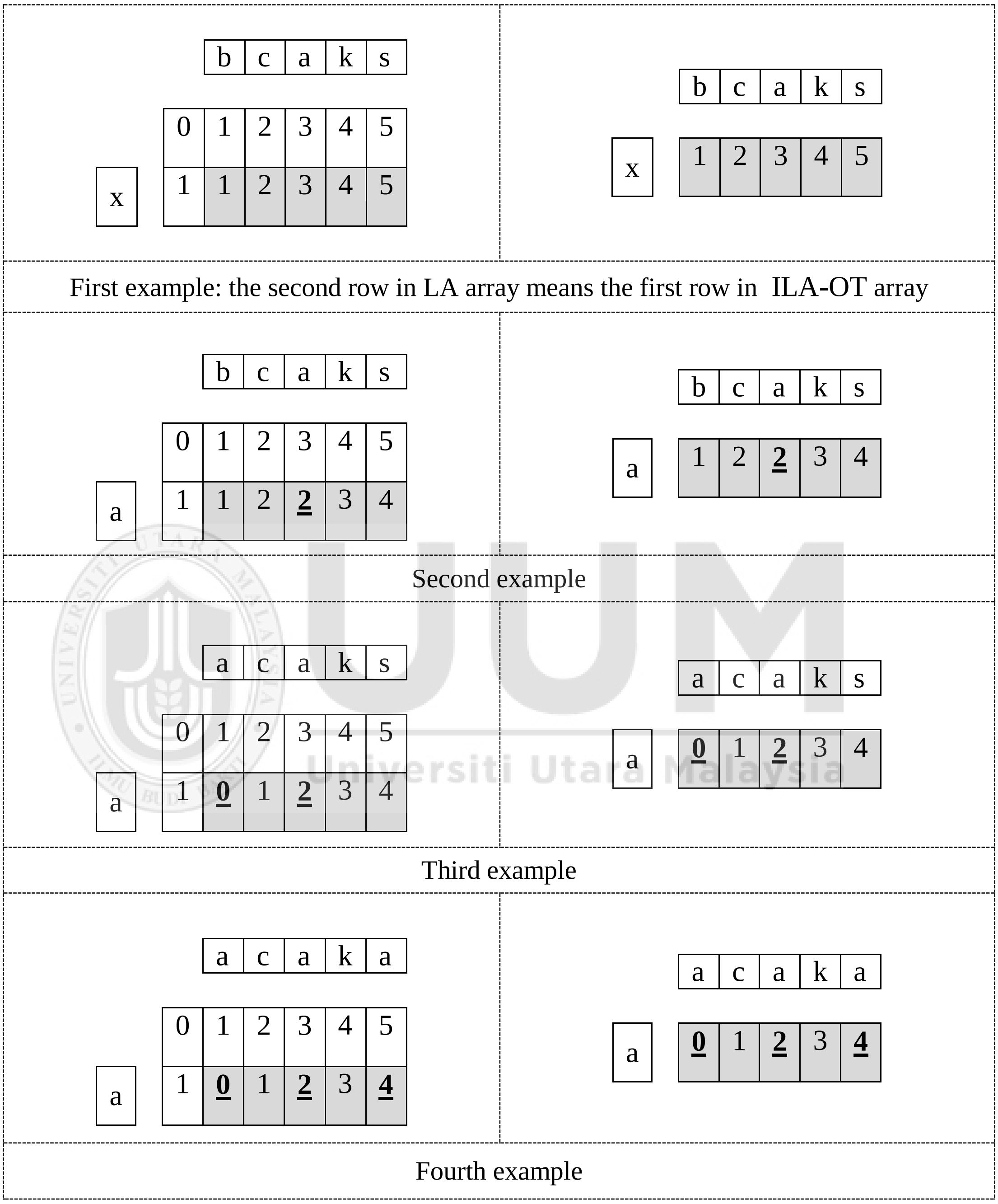
Figure 4.1 shows an example of how to calculate the distance between two words “back” and “book” using LA. It shows that first step of the proposed operational technique removes first row and first column from Levenshtein array as presented in part B. This is because values of the first row and first column are used to measure values of second row and second column in Levenshtein array. Therefore, measuring values of second row and second column directly without needed to the values of the first row and the first column leads ignoring them in Levenshtein array. Figure 4.1 also shows that number of cells in LA array before applying the first rule is 25 and after applying this rule is 16. The percentage decrease in a number of cells of LA array is 36%. The percentage decrease has been calculated using Equation 4.1 (Lund, 2014).

$$\text{Percentage decrease} = \frac{\text{Old value - new value}}{\text{Old value}} * 100 \qquad (4.1)$$

Where the term “old value” represents a number of cells before applying the first step, which is 25, while the term “new value” represents the number of cells after applying the first step, which is 16. Another example, if a number of characters of input two words to the LA is 5 and 6, then number of cells in LA array is 42 ((5+1) * (6+1)) while after applying the first step is only 30 (5 * 6). The percentage decrease in a number of cells of LA array for the second example is 28.57%.

4.2.2 Second Step

Starting from this section, this study has been referred to the second row and second column as the first row and first column in the ILA-OT array. This is due to the removing first row and first column from Levenshtein array after applying the first step as shown in Figure 4.1. The second step of the proposed operational technique can be applied to measure cells values of the first row in the ILA-OT array (second row in LA array) and first column in the ILA-OT array (second column in LA array). This step requires fewer operations than others performing in LA. Figure 4.2 shows four examples that explain second step implementation by comparing values of the second row in LA array with values of the first row in ILA-OT array.



		b	c	a	k	s
	0	1	2	3	4	5
x	1	1	2	3	4	5

	b	c	a	k	s
x	1	2	3	4	5

First example: the second row in LA array means the first row in ILA-OT array

		b	c	a	k	s
	0	1	2	3	4	5
a	1	1	2	**2**	3	4

	b	c	a	k	s
a	1	2	**2**	3	4

Second example

		a	c	a	k	s
	0	1	2	3	4	5
a	1	**0**	1	**2**	3	4

	a	c	a	k	s
a	**0**	1	**2**	3	4

Third example

		a	c	a	k	a
	0	1	2	3	4	5
a	1	**0**	1	**2**	3	**4**

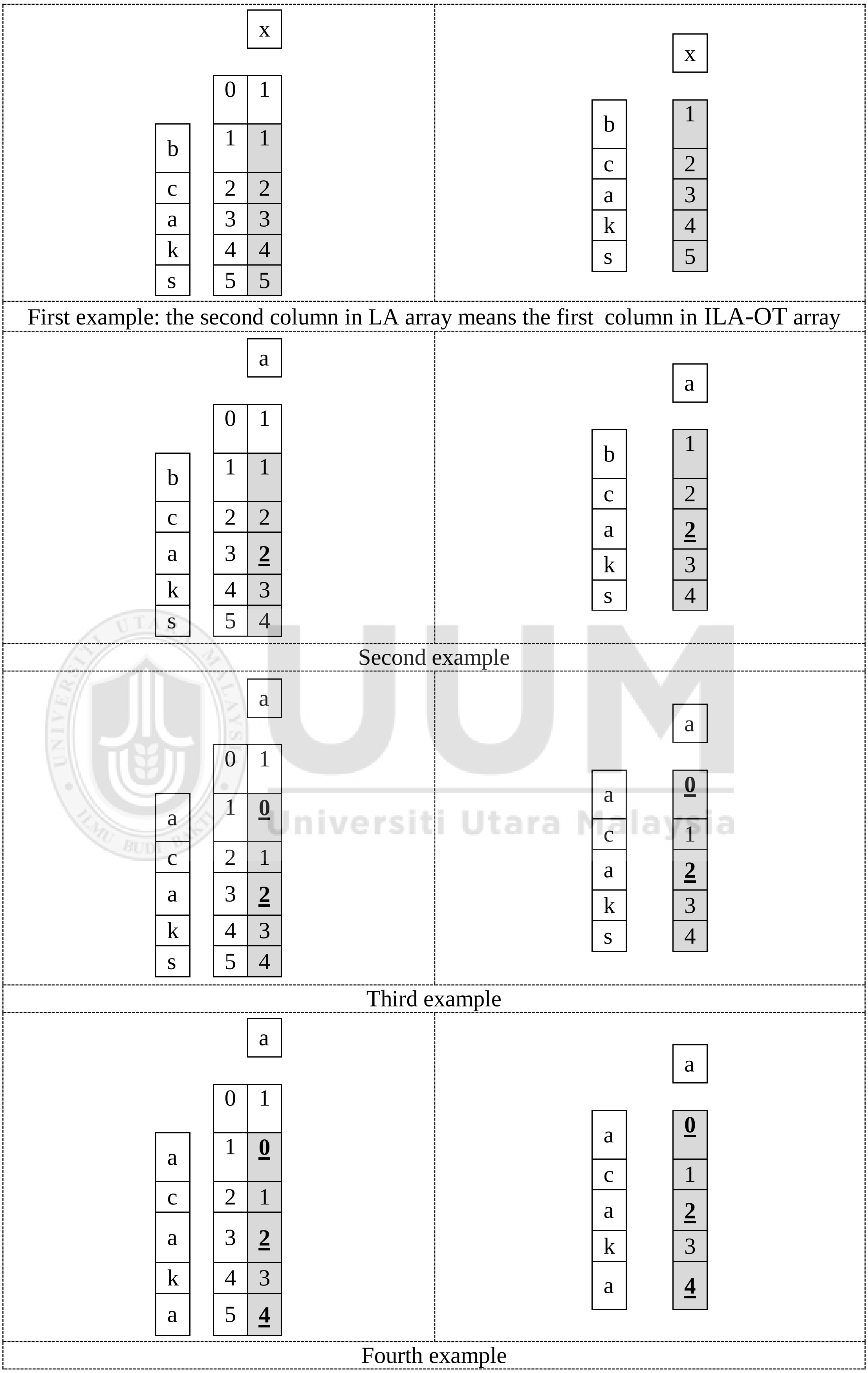
	a	c	a	k	a
a	**0**	1	**2**	3	**4**

Fourth example

Figure 4.2. Examples of the second step for the first row

This dissertation uses the term d(i, j) to refer to any cell in ILA-OT array, where the character “i” represents the position of a row in ILA-OT array and in the same time it represents the position of each character in second word “t”, while the character “j” represents the position of a column in ILA-OT array and in the same time it represents the position of each character in first word “s”. The first example in Figure 4.2 shows that if there is no shared character between characters of first-word “s(j)” and the first character in second-word t(0), then all cells of the first row will take the values of the context (1+j) in the ILA-OT array. Note the order of characters in any string is 0 for the first character, 1 for the second character, 2 for the third character, and so on. The examples of second, third, and fourth show that if there is a shared character, then all cells values of the first row in the ILA-OT array, starting from the cell that has shared a character, will take the value of “j” until the last cell in the first row without needed to extra comparing. This is because first shared character will make shared cell value equal to the “j” instead of (j+1). Furthermore, the context will be constant and it will not be changed if there is an additional shared character in the same row.

Figure 4.2 also shows that if the proposed second step is performed, the values of the second row in LA array are matched with the values of the first row in ILA-OT array as shown in all examples in this figure. However, the operations required to measure cells values in the first row of ILA-OT is less than operations required to measure cells values in the second row in LA for the reasons mentioned in Section 4.3. The second step of the proposed operational technique can also be applied to measure cells values of the first column in the ILA-OT array (second column in LA array) with a simple modification. The modification is to replace “j” by “i” as shown in the examples of Figure 4.3.



First example: the second column in LA array means the first column in ILA-OT array

Second example

Third example

Fourth example

Figure 4.3. Examples of the second step for the first column

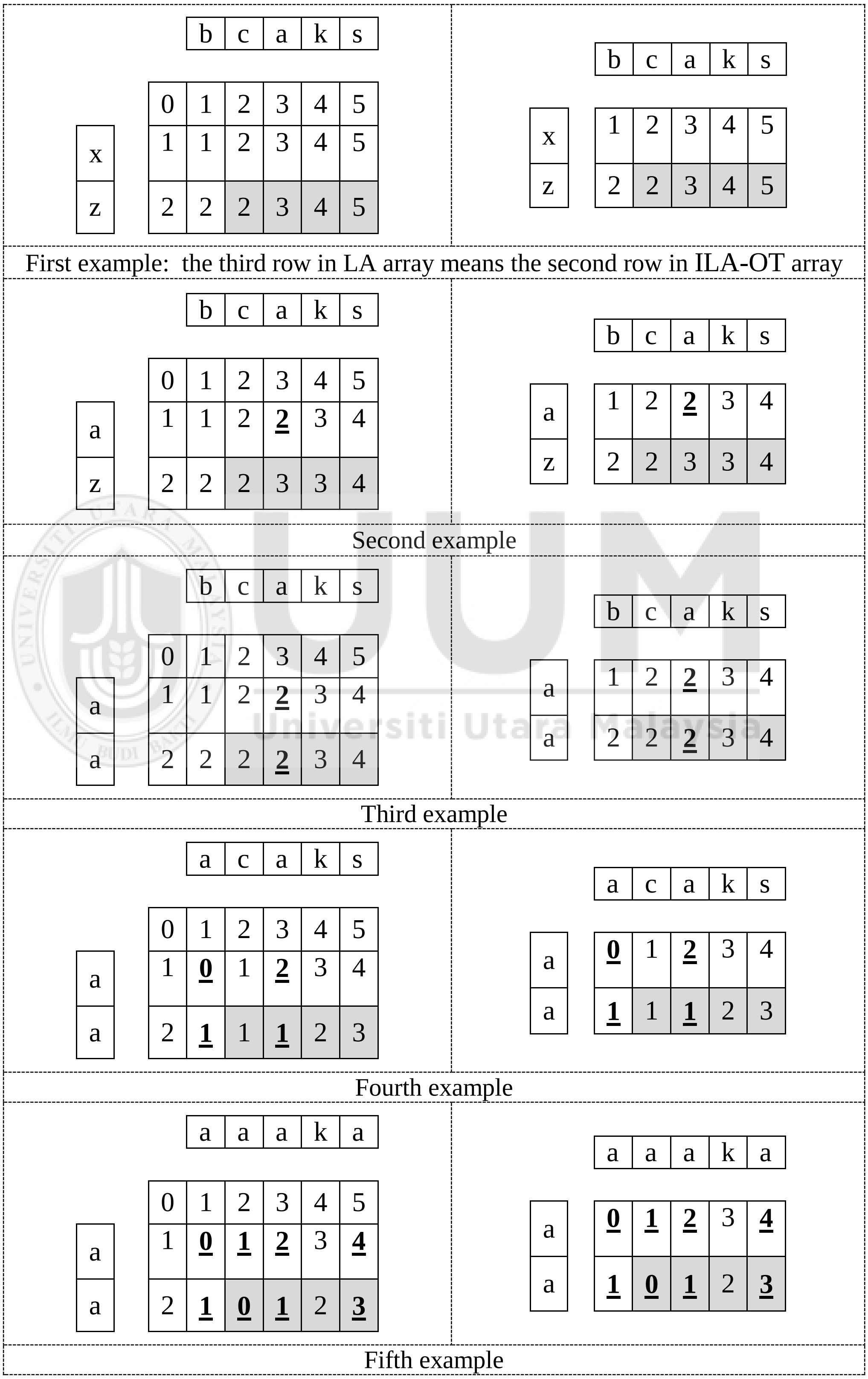
The mathematical expressions of the second step of the proposed operational technique for both first row and first column in the ILA-OT array are discussed in Section 4.3. Furthermore, the details of implementing the second step are presented as an algorithm in Section 4.4. Finally, the second step of the proposed operational technique can be summarized as two sentences are written below:

1) Each cell in the first row of ILA-OT array will take the value of the context (1+j) until finding shared character. Starting from the cell that has shared character until the last cell in the first row, the cell value will become j without needed to extra compare.

2) Each cell in the first column of ILA-OT array will take the value of the context (1+i) until finding shared character. Starting from the cell that has shared character until the last cell in the first column, the cell value will become "i" without needed to extra comparing.

4.2.3 Third Step

This step can be applied to measure cells values of the second row in ILA-OT array (third row in LA array) and second column in the ILA-OT array (third column in LA array). The third step also finds a context in measuring values of second row and second column in ILA-OT array. However, the context of the second step is different from the context of the third step. The context of the third step needs values of first row and first column to measure values of second row and second column while the context of the second step does not need values from any row or column. The third step also requires fewer operations than others performing in original LA for the reasons mentioned in Section 4.3. Figure 4.4 shows five examples that explain third step implementation by comparing values of the third row in LA array with values of the second row in ILA-OT array.



		b	c	a	k	s
	0	1	2	3	4	5
x	1	1	2	3	4	5
z	2	2	2	3	4	5

	b	c	a	k	s
x	1	2	3	4	5
z	2	2	3	4	5

First example: the third row in LA array means the second row in ILA-OT array

		b	c	a	k	s
	0	1	2	3	4	5
a	1	1	2	**2**	3	4
z	2	2	2	3	3	4

	b	c	a	k	s
a	1	2	**2**	3	4
z	2	2	3	3	4

Second example

		b	c	a	k	s
	0	1	2	3	4	5
a	1	1	2	**2**	3	4
a	2	2	2	**2**	3	4

	b	c	a	k	s
a	1	2	**2**	3	4
a	2	2	**2**	3	4

Third example

		a	c	a	k	s
	0	1	2	3	4	5
a	1	**0**	1	**2**	3	4
a	2	**1**	1	**1**	2	3

	a	c	a	k	s
a	**0**	1	**2**	3	4
a	**1**	1	**1**	2	3

Fourth example

		a	a	a	k	a
	0	1	2	3	4	5
a	1	**0**	**1**	**2**	3	**4**
a	2	**1**	**0**	**1**	2	**3**

	a	a	a	k	a
a	**0**	**1**	**2**	3	**4**
a	**1**	**0**	**1**	2	**3**

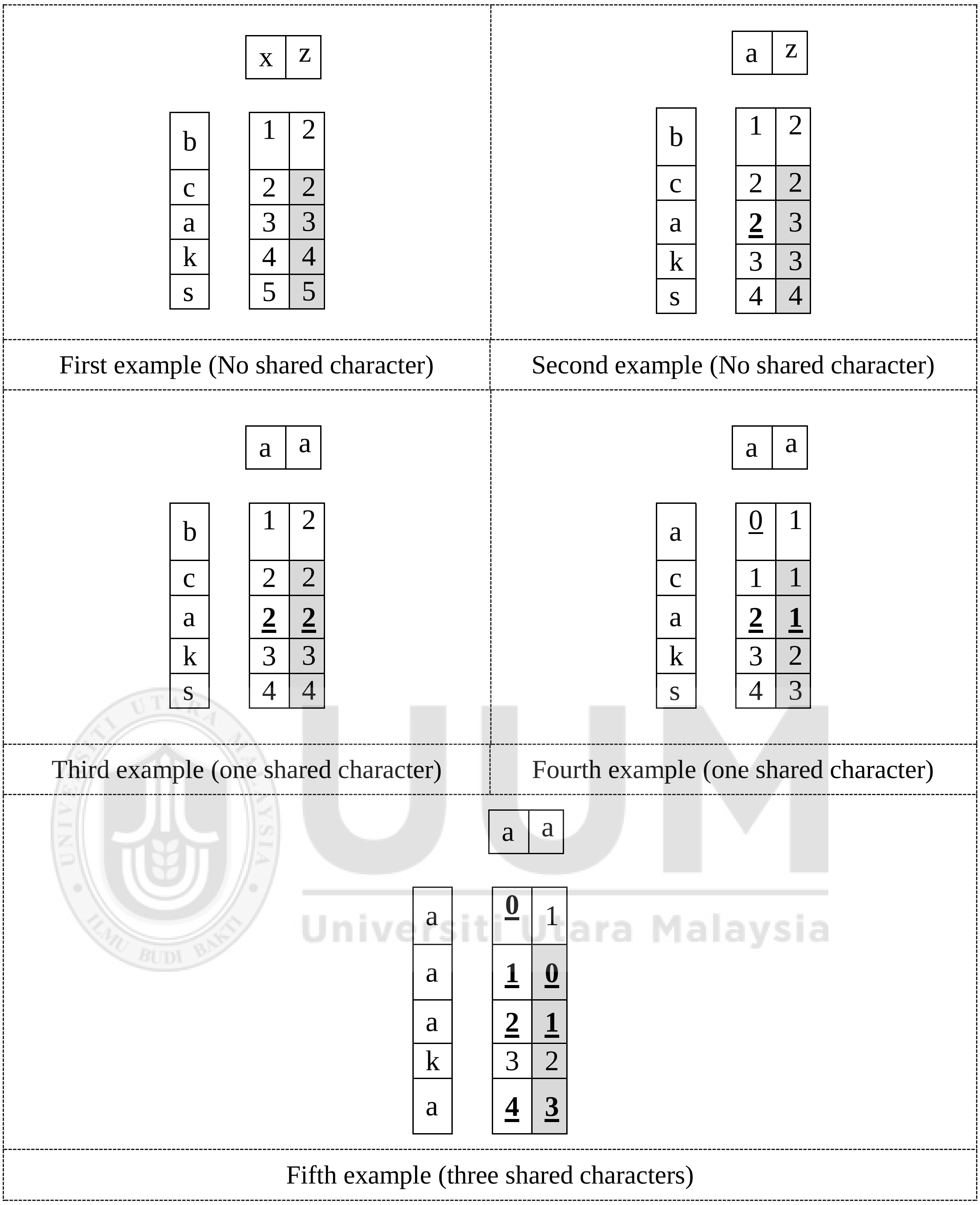
Fifth example

Figure 4.4. Examples of the third step for the second row

Examples one and two in Figure 4.4 shows that if there is no shared character between characters of first token “s(i)” and second character in second token t(1), then all values of the second row will take the values of the context: (1+d(0, j-1)). The examples of third, fourth, and five show that if there is a shared character, then the shared cell will takes the value of d(0, j-1), while all cells values of the second row, starting from the cell that follows shared cell, will take the value of (1+d(1, j-1)) until the last cell in the second row without needed to extra comparing. Note the context (1+d(1, j-1)) will be constant and it will not be changed if there is an additional shared character in the same row.

Figure 4.4 also shows that if the third step of the proposed operational technique is performed, the values of the third row in LA array are matched with the values of the second row in ILA-OT array. However, the operations required to measure cells values in the second row of ILA-OT is less than operations required to measure cells values in the third row in LA for the reasons mentioned in Section 4.3. The details of implementing the third step are presented as an algorithm in Section 4.4.

The third step of the proposed operational technique can also be applied to measure cells values of the second column in the ILA-OT array (third column in LA array) with a simple modification. The modification is to replace “j” by “i” as shown in the examples of Figure 4.5.



	x	z
b	1	2
c	2	2
a	3	3
k	4	4
s	5	5

First example (No shared character)

	a	z
b	1	2
c	2	2
a	**2**	3
k	3	3
s	4	4

Second example (No shared character)

	a	a
b	1	2
c	2	2
a	**2**	**2**
k	3	3
s	4	4

Third example (one shared character)

	a	a
a	0	1
c	1	1
a	**2**	**1**
k	3	2
s	4	3

Fourth example (one shared character)

	a	a
a	**0**	1
a	**1**	**0**
a	**2**	**1**
k	3	2
a	**4**	**3**

Fifth example (three shared characters)

Figure 4.5. Examples of the third step for the second column

The mathematical expressions of the third step for both second row and second column in the ILA-OT array are discussed in Section 4.3. Finally, the third step can be summarized as two sentences are written below:

1) Each cell in the second row of ILA-OT array will take the value of the context (1+d(0, j-1)) until finding shared character. The shared cell will take the value of d(0, j-1), while all cells values of the second row, starting from the cell that follows shared cell, will take the value of (1+d(1, j-1)) without needed to extra comparing.

2) Each cell in the second column of ILA-OT array will take the value of the context (1+d(i-1,0)) until finding shared character. The shared cell will take the value of d(i-1,0), while all cells values of the second column, starting from the cell that follows shared cell, will take the value of (1+d(i-1, 1)) without needed to extra compare.

4.3 Comparison between operations of LA and ILA-OT

In this section, a comparison between LA and ILA-OT regarding the operations required to measure each cell value in their arrays is presented. The comparison is based on mathematical expression of both LA and ILA-OT. Equation 4.2, Equation 4.3, Equation 4.4, and Equation 4.5 shows the mathematical expression of LA, which have been extracted from LA algorithm presented in (Lounis et al., 2014).

$$\mathrm{d}(0, \mathrm{j}) = \mathrm{j} \quad \text{for LA} \tag{4.2}$$

$$\mathrm{d}(\mathrm{i}, 0) = \mathrm{i} \quad \text{for LA} \tag{4.3}$$

$$\cos \mathrm{t} = \begin{cases} 0 & ,\text{if } \mathrm{t(i)} = \mathrm{s(j)} \\ 1 & ,\ \text{if } \mathrm{t(i)} \neq \mathrm{s(j)} \end{cases} \tag{4.4}$$

$$\mathrm{d(i, j)} \text{ for LA} = \min \begin{cases} 1 + \mathrm{d(i, j\text{-}1)} \\ 1 + \mathrm{d(i\text{-}1, j)} \\ \mathrm{cost} + \mathrm{d(i\text{-}1, j\text{-}1)} \end{cases} \tag{4.5}$$

Where the term d(i, j) represents the array required to measure edit distance, while the symbol "i" represents the position of a row in LA array and in the same time it represents the position of each character in second word "t", while the character "j" represents the position of a column in LA array and in the same time it represents the position of each character in first word "s". Equation 4.2 and Equation 4.3 show how can measure each cell value in the first row and first column respectively for LA array while Equation 4.4 and Equation 4.5 show how can measure each cell value for other rows and columns in LA array.

Equation 4.6, Equation 4.7, Equation 4.8 and Equation 4.9 show how can measure each cell value in the first row, first column, second row, and second column respectively for the ILA-OT array.

$$d(0, j) = \begin{cases} j+1 & \text{, before } t(0) = s(j) \\ j & \text{, if } t(0) = s(j) \text{ or after this condition} \end{cases} \tag{4.6}$$

$$d(i, 0) = \begin{cases} i+1 & \text{, before } t(i) = s(0) \\ i & \text{, if } t(i) = s(0) \text{ or after this condition} \end{cases} \tag{4.7}$$

$$d(1, j) = \begin{cases} 1 + d(0, j-1) & \text{, before } t(1) = s(j) \\ d(0, j-1) & \text{, if } t(1) = s(j) \\ 1 + d(1, j-1) & \text{, after } t(1) = s(j) \end{cases} \tag{4.8}$$

$$d(i, 1) = \begin{cases} 1 + d(i-1, 0) & \text{, before } t(i) = s(1) \\ d(i-1, 0) & \text{, if } t(i) = s(1) \\ 1 + d(i-1, 1) & \text{, after } t(i) = s(1) \end{cases} \tag{4.9}$$

Where the term d(i, j) represents the array required to measure edit distance, while the symbol "i" represents the position of a row in ILA-OT array and in the same time it represents the position of each character in second word "t", while the character "j" represents the position of a column in ILA-OT array and in the same time it represents the position of each character in first word “s”. Also, as mentioned previously, second row and second column in LA array mean first row and first column in ILA-OT array, while third row and third column in LA array mean second row and second column in the ILA-OT array. This is due to the removing first row and first column from the ILA-OT array as described in Section 4.2.1.

Before comparing the operations of both algorithms LA and ILA-OT, this dissertation ignores increasing operations required to increase “i” and “j” from the calculation because they are almost same for ILA-OT array and LA array. Furthermore, it ignores any single operation does not require a loop to measure it in order to make the comparison of operations easier.

Equation 4.2 and Equation 4.3 show that it needs one assigning operation to fill each cell value in the first row and first column of LA array. To fill other rows and columns in LA array, it requires measuring equation 4.4 and equation 4.5. To identify the value of cost in equation 4.4, it needs two operations: one comparing operation and one assigning operation, while to identify the value of d(i, j) in equation 4.5, it needs six operations: three summing operations, two comparing operations and one assigning operation. The total number of operations is 8 for each cell in LA array excluding first row and first column.

On the other hand, Equation 4.6 and Equation 4.7 show that each cell value in the first row and first column in the ILA-OT array (second row and second column in LA array) requires three operations: one summing operation, one comparing operation and one assigning operation. Furthermore, it requires fewer operations if there is a shared character between characters of first token “s(j)” and the first character in second token t(0). This means the proposed technique decreases 5 operations from 8 operations needed in a normal way to measure each cell value in the first row and first column.

On the other hand, Equation 4.8 and Equation 4.9 show that each cell in the second row and second column in the ILA-OT array (third row and third column in LA array), also requires three operations: one summing operation, one comparing operation and one assigning operation. Furthermore, it also requires fewer operations if there is a shared character between characters of first token “s(j)” and second character in second token t(1). This means the proposed technique will also decrease 5 operations from 8 operations needed in a normal way to measure each cell value in the second row and second column. Table 4.1 shows the operations needed to fill both ILA-OT array and LA array.

Table 4.1

Theoretical comparison between operations of LA and ILA-OT

	Cell operations in LA array	**Cell operations in ILA-OT array**
First row	1 * (number of cells in the first row)	0 because the first row is deleted
First column	1 * (number of cells in the first column)	0 because the first column is deleted
Second row	8 * (number of cells in the second row)	3 or less * (number of cells in the first row)
Second column	8 * (number of cells in the second column)	3 or less * (number of cells in the first column)

Third row	8 * (number of cells in the third row)	3 or less * (number of cells in the second row)
Third column	8 * (number of cells in the third column)	3 or less * (number of cells in the second column)
Other rows and columns	Same operations	Same operations

From Table 4.1, it can be seen that ILA-OT decreases the operations of cells in LA array in six positions. The first position is to remove all cells of the first row in LA array with their operations while the second position is to remove all cells of the first column in LA array with their operations. The third position is to decrease operations of the second row in LA array by almost (5 * number of cells in the second row) while the fourth position is to decrease operations of the second column in LA array by almost (5 * number of cells in the second column). The fifth position is to decrease operations of the third row in LA array by almost (5 * number of cells in the third row) while the sixth position is to decrease operations of the third column in LA array by almost (5 * number of cells in the third column).

As a practical example, if the number of characters in the first and second words is 5, then a number of cells in LA array is 6*6=36 while in ILA-OT array is 5*5=25. Furthermore, the approximate number of operations of LA array and ILA-OT array are shown in Table 4.2 on the next page.

Table 4.2

Practical comparison between LA and ILA-OT

	Cell operations in LA array	**Cell operations in ILA-OT array**
First row	1 * 6=6	0
First column	1 * 6=6	0
Second row	8 * 5=40	3 or less * 5=15 or less
Second column	8 * 5=40	3 or less * 5=15 or less
Third row	8 * 4=32	3 or less * 4=12 or less
Third column	8 * 4=32	3 or less * 4=12 or less
Other rows and columns	Same operations	Same operations

The comparisons in Table 4.1 and Table 4.2 show that the operations needed to fill ILA-OT array are less than operations needed to fill LA array. Finally, since an average word length for the English language is 5 characters (Lund, 2014), and by assuming the symbols c1, c2, c3, c4, and c5 refer to the characters in English word. Then 27 cells from 36 cells will be affected by three steps proposed by this study as shown in Figure 4.6. Note gray cells in Figure 4.6 refer to the cells that will be affected by three steps proposed by this study while white cells will be measured normally.

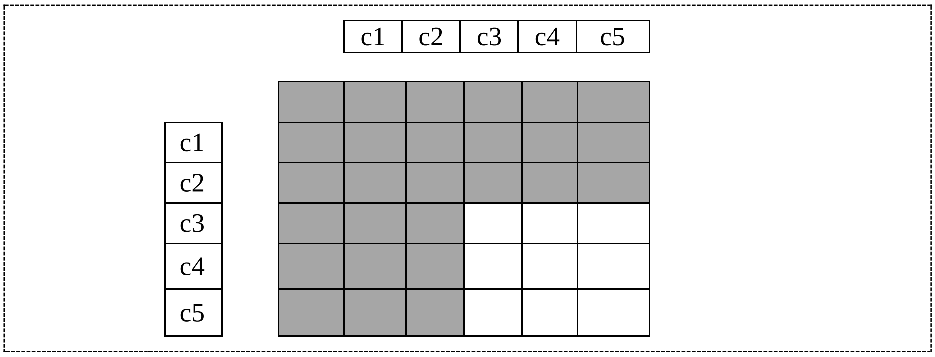


Figure 4.6. Cells affected by three steps proposed of this study

4.4 ILA-OT Algorithm

Algorithm 4.1 shows the pseudo-code of the improved Levenshtein algorithm (ILA-OT).

Algorithm 4.1: Improved Levenshtein algorithm (ILA-OT)

```
integer ILA-OT (string t, string s)
// i and j iterate over t and s
Integer i, j, cost, n, m
n= length(t)
m= length(s)
// First Step: remove a single row and single column from ILA-OT array
n=n-1
m=m-1
// d is an array of (n+1) rows and (m+1) columns
Integer d[0..n, 0..m]
// Second Step: measure values of the first row and first column using sub-algorithm named "MVFRFC". The algorithm is described in Algorithm 4.2
d = MVFRFC (d, t, s, n, m)
// Third Step: measure values of the second row and second column using sub-algorithm named "MVSRSC". The algorithm is described in Algorithm 4.3
d = MVSRSC (d, t, s, n, m)
// Other rows and columns: Measure values of other rows and other columns in the  ILA-OT array
for i from 2 to n:
    for j from 2 to m:
        if t[i] = s[j] then cost=0
        else cost := 1
        d[i, j] = minimum(
                                d[i-1, j] + 1,          // deletion
                                d[i, j-1] + 1,          // insertion
                                d[i-1, j-1] + cost      // substitution
                          )
    next j
```

Proposed operational technique (OT)

```
next i
return d[n, m]
End
```

To make Algorithm 4.1 easier to understand, this dissertation refers to the string “chaine1”, string “chaine2”, variable “lengthChaine1”, and variable “lengthChaine2” in Algorithm 2.1 of LA as “t”, “s”, “n”, “m” in Algorithm 4.1 of ILA-OT respectively. The ILA-OT algorithm receives two strings as inputs “s” and “t”. The variables m and n hold lengths of s and t respectively. The first step in the proposed operational technique (OT) is to decrease both m and n each by one. This is because proposed operational technique (OT) of ILA-OT algorithm removes a single row and single column from ILA-OT array as described in Section 4.2.1. Next, a two-dimensional array named “*d*” is created. This array has been used to measure edit distance between *“s”* and *“t”*.

After that, the second step in the OT is performed. This step measures cells’ values of the first row and first column using sub-algorithm named “MVFRFC”. The pseudo-code of the MVFRFC algorithm was presented in the Algorithm 4.2. The third step in the OT is to measure cells’ values of the second row and the second column using another sub-algorithm named “MVSRSC”. The pseudo-code of the MVSRSC algorithm was presented in the Algorithm 4.3. Both sub-algorithms MVFRFC and MVSRSC receive five parameters: d, t, s, n, and m. These parameters have been defined in the Algorithm 4.1.

To refer to any cell in the d array, two variables are used named “i” and “j”. The variable “i” refers to the position of the row in d array while variable “j” refers to the position of the column in the d array. Furthermore, the variable “j” also represents the position of each character in the first string “s”, while the variable “i” also represents the position of each character in the second string “t”. The last step in the ILA-OT algorithm is to fill other cells values in d array by using two loops. After that, the edit distance between strings “s” and “t” is located in the last cell in the d array, which is represented by d (n, m). Return to the sub-algorithm MVFRFC, the pseudo-code of it is shown in Algorithm 4.2 below.

Algorithm 4.2: MVFRFC

```
1   Input: d, t, s, n, m       // these inputs are taken from main Algorithm 4.1
    // Output: d
    // First part (S1): measure values of the first row
2       for j from 0 to m:
3           if s[j] = t[0]
4               for k from j to m:
5                   d(0, k) = k
6               next k
7               goto step 12
8           else
9               d(0, j) = j+1
10          end if
11      next j
    // Second part (S2): measure values of first column
12      for i from 0 to n:
13          if s[0] = t[i]
14              for k from i to m:
15                  d(k, 0) = k
16              next k
17              goto step 22
18          else
19              d(i, 0) = i+1
20          end if
21      next i
```

```
Return d
End
```

Algorithm 4.2 contains two parts S1 and S2. The part S1 uses to measure each cell value in the first row of array d while part S2 uses to measure each cell value in the first column of array d. In part S1, the MVFRFC algorithm starts by assigning 0 to the variable “j”. Next, one of the two cases will occur based on specific condition. The condition will examine if there is a shared character between characters of first string “s(j)” and the first character in second string t(0). If there is no shared character, then MVFRFC algorithm assigns for each cell value in the first row the context (j+1) until the last cell in the first row.

Otherwise, if a shared character is found, then all cells values of the first row, starting from the cell that has shared a character, take the value of “j” until the last cell in the first row without needed to the extra compare operation. Note the variable “j” is replaced by the variable “k” because it cannot use two counters with the same name in nested two loops in the same algorithm. In part S2, the MVFRFC algorithm applies same steps performed in S1 to measure cells values of the first column in the ILA-OT array with a simple modification. The modification is to replace “j” by “i” in order to refer to the cell in the first column instead of the first row.

Return to the sub-algorithm MVSRSC, the pseudo-code of it is shown in Algorithm 4.3.

Algorithm 4.3: MVSRSC

```
Input: d, t, s, n, m       // these inputs are taken from main Algorithm 4.1
// Output: d
```

```
    // First part (X1): measure values of the second row
2       for j from 1 to m:
3           if s[j] = t[1]
4               d(1, j) = d(0, j-1)
5               for k from (j+1) to m:
6                   d(1, k) = 1+ d(1, k-1)
7               next k
8               goto step 13
9           else
10              d(1, j) = 1+ d(0, j-1)
11          end if
12      next j
    // Second part (X2): measure values of second column
13      for i from 1 to n:
14          if s[1] = t[i]
15              d(i, 1) = d(i-1, 0)
16              for k from (i+1) to n:
17                  d(k, 1) = 1+d(k-1, 1)
18              next k
19              goto step 24
20          else
21              d(i, 1) = 1+d(i-1, 0)
22          end if
23      next i
24  Return d
25  End
```

Algorithm 4.3 also contains two parts called X1 and X2. The part X1 uses to measure each cell value in the second row of array d while part X2 uses to measure each cell value in the second column of array d. In part X1, the MVSRSC algorithm starts by assigning 1 to the variable “j”. Next, one of the two cases will occur based on specific condition. The condition will examine if there is a shared character between characters of second string “s(j)” and the second character in second string t(1). If there is no shared character, then MVSRSC algorithm assigns for each cell value in the second row the context (1+ d(0, j-1)) until the last cell in the second row.

Otherwise, if a shared character is found between both strings, then the shared cell takes the value of d(0, j-1), while other cells values of the second row, starting from the cell that follows the shared cell, take the value of (1+ d(1, j-1)) until the last cell in the second row without needed to extra compare operation. In part X2, the MVSRSC algorithm applies same steps performed in X1 to measure cells values of the second column in an ILA-OT array with a simple modification. The modification is to replace “j” by “i” in order to refer to the cell in the second column instead of the second row. Lastly, as mentioned in Algorithm 4.2, the variable “j” is also replaced by the variable “k” because it cannot use two counters with the same name in nested two loops in the same algorithm.

4.5 Summary

In this chapter the detail regarding the improved Levenshtein algorithm i.e. the ILA-OT is presented and discussed in detail. The discussion begins with the steps and notation of proposed operational technique which is the core of the algorithm proposed in this dissertation. Examples are provided for each step to show how ILA-OT measures cells values in its array. This is followed by algorithms to implement ILA-OT. The comparison between LA and ILA-OT regarding the operations required to measure each cell value in their arrays is presented. Furthermore, a practical example of the comparison process has also been presented. As can be seen in this chapter the proposed operational technique is based on the idea of finding a context to measure cells values in LA with fewer operations than what required in the traditional way.

CHAPTER FIVE
EXPERIMENTAL RESULTS AND DISCUSSION

5.1 Introduction

This chapter discussed the experimental results of the improved Levenshtein algorithm (ILA-OT) using single testing dataset containing various word lengths. The experimental design, evaluation measurements, and testing dataset were explained in Chapter Three. Six sections have been included in this chapter. Section 5.2 presents the experimental results that related to the processing time. This is followed by Section 5.3 that presents the experimental results that related to the accuracy. Section 5.4 shows the results of the statistical test conducted on processing time values of both LA and ILA-OT. A discussion on results obtained by this chapter is explained in Section 5.5. Finally, a summary of chapter five is presented in Section 5.6

5.2 Processing Time

Table 5.1 below presents a sample of the processing time values of both algorithms for several words.

Table 5.1
A sample of the processing time values of both algorithms for several words

Word length	**Number of tests**	**Input**		**Output**	
		Word1	**Word2**	**Average processing time in microsecond (LA)**	**Average processing time in microsecond (ILA-OT)**
3	5	cab	bad	0.72	0.48

4	5	cozy	zyme	1.14	0.72
5	5	abase	abash	1.54	1.06
6	5	abacus	abaist	2.20	1.46
7	5	aaronic	abacist	2.70	2.04
8	5	aalsmeer	aaltonen	3.44	2.46
9	5	abanooneo	abarbanel	4.18	3.02
10	5	abandonmen	abatements	5.16	3.76
11	5	abbotsholme	abbreviated	6.06	4.50
12	5	abbreviatons	abbreviators	6.88	5.16

Table 5.1 shows that this dissertation conducted five tests for each input. The input of each algorithm is two words and the five tests have the same input words. After that, the average processing time for five tests has been measured. Average processing time for LA has been measured using the step 7 to step 15 in Algorithm 3.1 while average processing time for ILA-OT has been measured using the step 16 to step 24 in the same algorithm. Table 5.1 also shows that the processing time values of both algorithms are different for the words used in this table. Furthermore, it shows that the processing time values of ILA-OT are less than processing time values of LA for all lengths.

Appendix A to Appendix J presents samples of the processing time values of both algorithms for each word length. Table 5.2 presents the experimental results of both algorithms LA and ILA-OT in terms of processing time for all words in the testing dataset.

Table 5.2

Processing times of both algorithms LA and ILA-OT

Word length	**Number of tests**	**Average processing time in microsecond (LA)**	**Average processing time in microsecond (ILA-OT)**	**Percentage decrease (PD)**
3	1000	0.82	0.47	42.35%
4	1000	1.10	0.68	38.07%
5	1000	1.57	0.99	36.45%
6	1000	2.17	1.41	35.12%
7	1000	2.72	1.82	33.17%
8	1000	3.49	2.44	29.90%
9	1000	4.25	3.03	28.60%
10	1000	5.14	3.79	26.21%
11	1000	6.18	4.63	25.13%
12	1000	7.42	5.64	23.91%
All words (From 3 to 12)	10000	3.48	2.49	28.50%

Table 5.2 shows that this dissertation conducted 10000 tests for the words in the testing dataset (i.e. 1000 tests for each word length). In each test, the input to both algorithms is two words each time and the output is the processing time of execution of each algorithm. The inputs and outputs of all test are stored in excel file as explained in Algorithm 3.1. After that, average processing time values have been measured automatically for each length using Microsoft Excel 2010. Furthermore, average processing time for all lengths together has also been measured using Microsoft Excel 2010.

Table 5.2 also shows that the proposed algorithm ILA-OT has a percentage decrease for all word lengths. The average percentage decrease has been calculated using Equation 3.1 after implement it using Microsoft Excel 2010. The average percentage decrease obtained by the proposed algorithm for all lengths is 28.50% and for length 5, which represents average word length for the English language, is 36.45%. In addition to that, the experimental results also show that the impact of the proposed algorithm increases for short strings as presented in length 3, and it decreases for long strings as presented in length 12. Figure 5.1 shows the Line graph for the processing time values listed in this table.

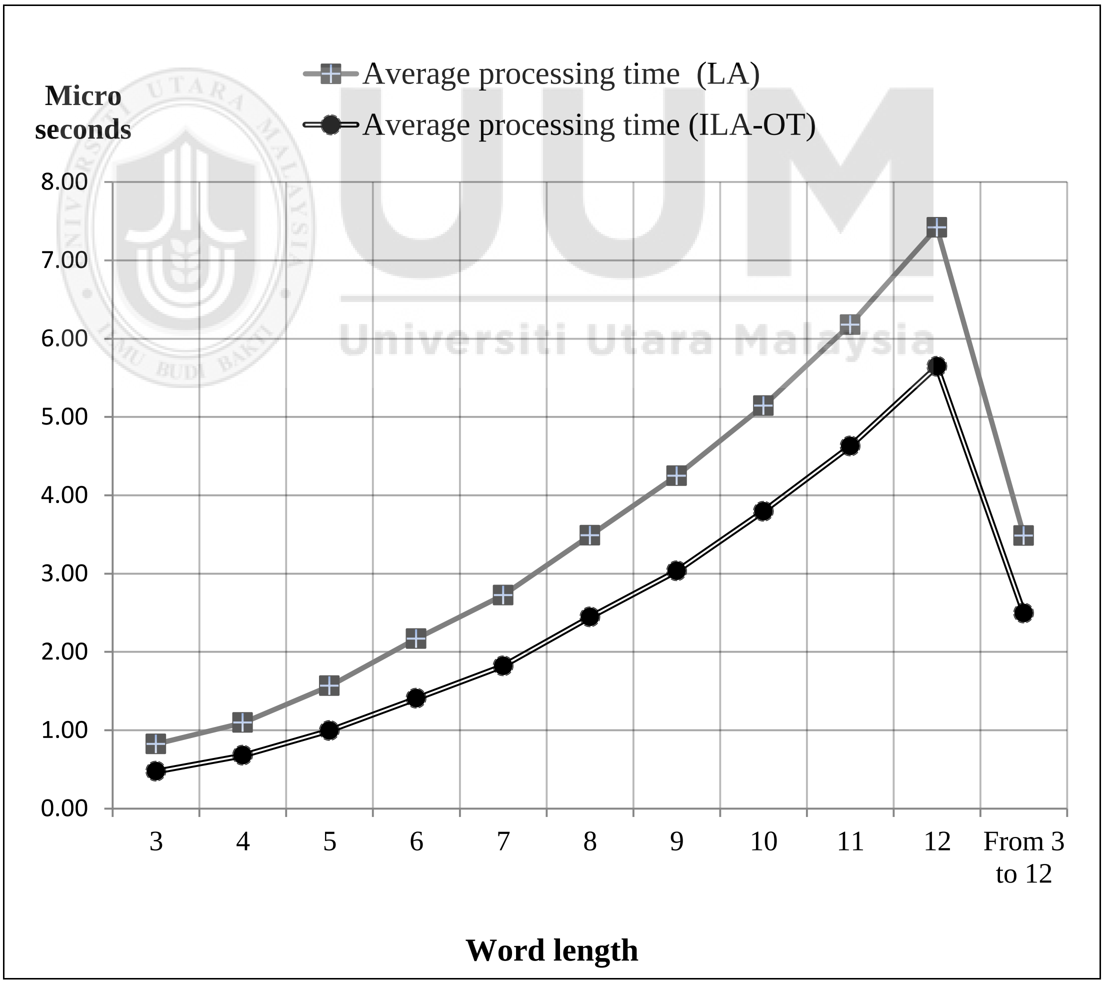


Figure 5.1. Line graph for the processing time values listed in Table 5.2

Figure 5.1 shows that the proposed algorithm ILA-OT outperformed the LA algorithm in terms of processing time for all lengths. However, the value of processing time for each length is different from the others due to the different number of characters in words used. Figure 5.1 also shows that the last values in Line graph are different from the others. This is because last values in the Line graph represent word lengths from length 3 to length 12. Lastly, as mentioned previously, since an average word length for the English language is 5 characters, then the proposed algorithm still has a significant decrease in processing time.

5.3 Accuracy

Most steps used to measure the accuracy are similar to the steps used in measuring processing time, which is conducting 10000 tests for the words in the testing dataset. However, the input to both algorithms is two words each time but the output is a distance of each algorithm. Table 5.3 below presents a sample of the input and output of both algorithms for several words.

Table 5.3

A sample of the distances of both algorithms for several words

Word length example	Input		Output	
	Word1	Word2	Distance (LA)	Distance (ILA-OT)
3	aba	baa	2	2
4	cozy	zyme	4	4
5	abase	abash	1	1
6	abacus	abaist	3	3
7	abidjan	abierta	4	4
8	abdovcit	abducens	5	5

9	aboutthis	abplanalp	7	7
10	abtificial	abundances	8	8
11	adversarial	adversaries	2	2
12	adventurists	adversatives	6	6

Table 5.3 shows that the outputs of both algorithms are equal for the words used in this table. Furthermore, for 10000 tests used in this experiment, the outputs also match. In addition to that, this dissertation performs additional 50000 tests and the outputs also same. Therefore, after applying Equation 3.2, the results indicates that the accuracy between both algorithms is 100% because the number of tests having equal distances between both algorithms is identical to the total number of the tests. Appendix A to Appendix J presents samples of the distance values of both algorithms for each word length.

5.4 Statistical Test

As mentioned previously, this dissertation performed a statistical method using a t-test to show if the improvement in processing time of the ILA-OT is significant or not. Table 5.4 below shows t-test results between LA and ILA-OT.

Table 5.4

T-test results between LA and ILA-OT

No.	**Main Variables**	**LA**	**ILA-OT**
1	Mean	3.48	2.49
2	Variance	7.04	3.29
3	Observations	10000	10000

4	t Stat	55.18
5	P(T<=t) two-tail	0.0
6	t Critical two-tail	1.96

Table 5.4 shows that the number of tests of each algorithm is 10000 for the words in the testing dataset, which represented a number of observations in t-test as shown in row 3. Table 5.4 also shows that the mean of ILA-OT is better than LA as is clear in row 1. Furthermore, it shows that the p-value is 0.0 as presented in row 5, which is less than 0.05, and the value of "t-Stat" is 55.18 as presented in row 4, which is much larger than the value of "t-critical" (1.96) in row 6. This indicates that the difference in means between LA and ILA-OT is real and not due to chance. Therefore, ILA-OT is better than LA in terms of processing time metric.

5.5 Results Discussion

The experimental results of this study show that the proposed algorithm ILA-OT outperformed the LA algorithm in terms of processing time for all word lengths. However, the processing time of both algorithms for each length is different from the others due to the different number of characters in words used. The experimental results also show that the impact of the proposed algorithm has been increased for short strings, and it has been decreased for long strings. However, as mentioned previously, since an average word length for the English language is 5 characters, then the proposed algorithm still has a significant decrease in processing time.

The average percentage decrease obtained by the proposed algorithm for all lengths is 28.50% and for length 5, which represents average word length for the English language, is 36.45%. For the accuracy metric, the results show that in all tests, which are 10000 comparisons, the number of tests having equal distances between both algorithms is identical to the total number of the tests. Therefore, the accuracy between both algorithms is 100% for all word lengths. Lastly, the experimental results of this chapter indicate that the objective one and two of this study have been achieved.

5.6 Summary

The goal of this chapter is to evaluate the proposed algorithm ILA-OT by comparing it with LA algorithm using two metrics, processing time and accuracy. Therefore, in this chapter, the details of experimental results are presented and discussed. The study conducted 10000 tests to evaluate this algorithm using 10 different word lengths. Furthermore, this dissertation conducted a statistical test to show if the improvement of the ILA-OT is significant or not. A statistical test of this dissertation has been measured using t-test. The results of the evaluation process are presented in detail. The experimental results are very encouraging. The proposed algorithm outperforms LA algorithm in terms of processing time without affecting its accuracy.

CHAPTER SIX
CONCLUSION AND FUTURE WORK

6.1 Introduction

This chapter finishes up the dissertation by making the conclusion for all chapters included in this research. The chapter begins with the research summary in Section 6.2 and then presents the main research contributions in Section 6.3. After that, the future directions of this work are provided in Section 6.4. Lastly, the chapter ends with a summary of this chapter in Section 6.5.

6.2 Research Summary

Similar to any other natural language processing problems, candidates' list generation in spelling correction is a real challenging problem which is very hard to tackle. These types of problems required exact and high-speed algorithms to achieve the intended goal. Each algorithm of candidates' list generation shows good performance for a specific purpose. Since the most widely used technique for generating candidates list for incorrect words is based on the Levenshtein algorithm, then this study has been focused on improving its processing time without affecting its accuracy.

The first step in developing improved algorithm is to design an operational technique to reduce the operations required to measure cells' values in Levenshtein array. This technique is based on the idea that it can remove first row and first column from LA array. Furthermore, it can predict cells' values in a second row, second column, third row, and third column in LA array without needed a traditional way. This is because there is a context to measure cells values in these rows and columns until finding a shared character in them. After finding a shared character, the context will be changed, therefore, it can measure cells values in these rows and columns based on the new context. Lastly, the context will only be changed once, and it will not change if there is an additional shared character in them.

Experiments were conducted to illustrate how the proposed improved Levenshtein algorithm can be employed to yield a promising result. The evaluation process includes comparing the proposed algorithm with LA algorithm using two metrics. The proposed algorithm has significantly reduced the processing time of measuring edit distance between any two strings compared to the current algorithm. Furthermore, the testing dataset that contains 10000 words used in the evaluation process proved its ability to not change the accuracy of the current algorithm. This is because both ILA-OT and LA produce same distance for all words in the testing dataset.

6.3 Research Contributions

This section summarizes the main contributions of the research by referring to the study objectives stated in Section 1.4 of Chapter one. The objectives are:

a. To design an operational technique to shorten the operations of the cells in the Levenshtein array without affecting its accuracy.

The first objective is to design an operational technique that decreases the operations required to measure each cell value in Levenshtein array without affecting its accuracy. Therefore, this research proposes an improved Levenshtein algorithm which is called ILA-OT (refer to the Algorithm 4.1, page 55) that contains the operational technique. The proposed technique (refer to the pages 39-49) is based on three steps. The first step removes first row and first column from LA array. Since an average word length for the English language is 5 characters, then the first step has a percentage decrease in a number of cells of LA array by 30.55% (refer to the page 40). The second and third steps can predict cells values in a second row, second column, third row, and third column in LA array with fewer operations than what required in a traditional way (refer to the page 49).

The second and third steps have decreases the operations of cells in LA array in four positions (refer to the Table 4.1, page 52). The first position is to decrease operations of the second row in LA array by almost (5 * number of cells in the second row) while the second position is to decrease operations of the second column in LA array by almost (5 * number of cells in the second column). The third position is to decrease operations of the third row in LA array by almost (5 * number of cells in the third row) while the fourth position is to decrease operations of the third column in LA array by almost (5 * number of cells in the third column).

b. To develop and evaluate the improved Levenshtein algorithm that includes the proposed operational technique.

The second objective has been achieved by developing and evaluating the proposed ILA-OT. This has been done by implementing it using VB.NET programming language and comparing it with LA algorithm using two metrics. Details of the experiments were presented and discussed in Chapter 5. The study conducted experiments to evaluate this algorithm using 10 different word lengths. The proposed algorithm outperforms LA algorithm in terms of processing time without affecting its accuracy. In terms of processing time value, the proposed algorithm ILA-OT enhanced over LA algorithm by 36.45%. In terms of accuracy value, both ILA-OT and LA produce same distance for any two strings. The accuracy between both algorithms is 100% for all words in the testing dataset.

6.4 Recommendation for Future Work

During the process of this research, several directions arose, which are considered as good seeds for future research. The following points highlight the promising directions found by this research:

i. The proposed algorithm ILA-OT has only decreased the operations of the first row, first column, second row, second column, third row, and third column in LA array. Therefore, it needs to include other rows and columns in LA array.

ii. The proposed algorithm ILA-OT could be explored in other applications that initially utilized Levenshtein algorithm and this includes DNA searching, sequences' alignment, word-processing programs, speech processing systems, and optical character recognition systems.

iii. Further research can be done to improve the proposed algorithm by designing a pre-processing technique that can help ILA-OT in filtering lexicon words.

iv. Finally, the hybridization between two or more edit distance algorithms may achieve better processing time than what is obtained in individual algorithm alone.

6.5 Summary

As a conclusion for this dissertation, this chapter has discussed and concluded the research summary, the contributions of the research. At the end of the discussion, the recommendations for further research were outlined. Overall, this dissertation has been achieved its main goal, which is to improve Levenshtein algorithm by reducing its processing time in generating candidates' list for spelling correction. Therefore, this study has designed an ILA-OT algorithm, which is based on the concept of finding patterns to use in predicting the cells' values in LA array instead of measuring values of them by using the traditional way of LA.

REFERENCES

Adhitama, P., Kim, S. H., & Na, I. S. (2014). Lexicon-Driven Word Recognition Based on Levenshtein Distance. *International Journal of Software Engineering and Its Applications, 8*(2), 11-20.

Ahmed, B. (2015). *Lexical Normalisation of Twitter Data.* Paper presented at the Science and Information Conference (SAI), 2015.

Al-Bakry, A. M., & Al-Rikaby, M. K. (2015). *Enhanced Levenshtein Edit Distance Method functioning as a String-to-String Similarity Measure.* Paper presented at the Proceedings of the Networks Security and Distributed Systems (NSDS'2015).

Al-Masoudi, A. F. R., & Al-Obeidi, H. S. R. (2015). Smoothing Techniques Evaluation of N-gram Language Model for Arabic OCR Post-processing. *Journal of Theoretical and Applied Information Technology, 82*(3), 432-439.

Al-Zaydi, Z. Q., & Salam, H. (2015). Multiple Outputs Techniques Evaluation for Arabic Character Recognition. *International Journal of Computer Techniques (IJCT), 2*(5), 1-7.

Alobaedy, M. M. T. (2015). *Hybrid Ant Colony System Algorithm For Static And Dynamic Job Scheduling In Grid Computing.* (PhD thesis, Universiti Utara Malaysia, Kedah, Malaysia). Retrieved from http://etd.uum.edu.my/id/eprint/5382

Andoni, A., & Krauthgamer, R. (2012). The smoothed complexity of edit distance. *ACM Transactions on Algorithms (TALG), 8*(4), 44.

Andoni, A., & Onak, K. (2012). Approximating edit distance in near-linear time. *SIAM Journal on Computing, 41*(6), 1635-1648.

Attia, M., Pecina, P., Samih, Y., Shaalan, K. F., & van Genabith, J. (2012). *Improved Spelling Error Detection and Correction for Arabic.* Paper presented at the International Conference on Computational Linguistics (COLING), Mumbai, India.

Bard, G. V. (2007). *Spelling-error tolerant, order-independent pass-phrases via the Damerau-Levenshtein string-edit distance metric.* Paper presented at the Proceedings of the fifth Australasian symposium on ACSW frontiers-Volume 68.

Bassil, Y., & Alwani, M. (2012a). Context-sensitive Spelling Correction Using Google Web 1T 5-Gram Information. *Computer and Information Science, 5*(3), 37-48.

Bassil, Y., & Alwani, M. (2012b). Ocr post-processing error correction algorithm using google online spelling suggestion. *Journal of Emerging Trends in Computing and Information Sciences, 3*(1), 90-99.

Batawi, Y., & Abulnaja, O. (2012). Accuracy Evaluation of Arabic Optical Character Recognition Voting Technique: Experimental Study. *IJECS: International Journal of Electrical & Computer Sciences, 12*(1), 29-33.

Bergroth, L., Hakonen, H., & Raita, T. (2000). *A survey of longest common subsequence algorithms.* Paper presented at the Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00).

Burkhardt, S., & Kärkkäinen, J. (2002). *One-gapped q-gram filters for Levenshtein distance.* Paper presented at the Proceedings of the Combinatorial Pattern Matching.

Cooper, L., & Cooper, M. W. (1981). *Introduction to dynamic programming*: Pergamon Press New York.

Crumrine, K. T., Ritschel, J. D., & White, E. (2014). Earned Schedule 10 Years Later Analyzing Military Programs: DTIC Document.

Daðason, J. F. (2012). *Post-Correction of Icelandic OCR Text.* (Master's thesis), University of Iceland, Reykjavik, Iceland.

Federico, M., & Cettolo, M. (2007). *Efficient handling of n-gram language models for statistical machine translation.* Paper presented at the Proceedings of the Second Workshop on Statistical Machine Translation.

Formiga Fanals, L., & Rodríguez Fonollosa, J. A. (2012). *Dealing with input noise in statistical machine translation.* Paper presented at the International Conference on Computational Linguistics (COLING), Mumbai, India.

Gomaa, W. H., & Fahmy, A. A. (2013). A survey of text similarity approaches. *International Journal of Computer Applications, 68*(13).

Google. (2015). Google Ngram Retrieved september 05, 2015, from http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

Grzebala, P. B. (2016). *Private Record Linkage: A Comparison of Selected Techniques for Name Matching.* Master thesis, Wright State University, Ohio, United States.

Gupta, B., Bhatt, G., & Mittal, A. (2016). Language Identification and Disambiguation in Indian Mixed-Script. *Distributed Computing and Internet Technology* (pp. 113-121): Springer.

Habeeb, I. Q., Yusof, S. A., & Ahmad, F. B. (2014). Two Bigrams Based Language Model for Auto Correction of Arabic OCR Errors. *International Journal of Digital Content Technology and its Applications, 8*(1), 72 - 80.

Haldar, R., & Mukhopadhyay, D. (2011). Levenshtein distance technique in dictionary lookup methods: An improved approach. *arXiv preprint arXiv:1101.1232*.

Hassan, A., Noeman, S., & Hassan, H. (2008). *Language Independent Text Correction using Finite State Automata.* Paper presented at the Third International Joint Conference on Natural Language Processing (IJCNLP), Hyderabad, India.

Huldén, M. (2009). Fast approximate string matching with finite automata. *Procesamiento del lenguaje natural*, *43*, 57-64.

Islam, A., & Inkpen, D. (2009). *Real-word spelling correction using Google Web 1T n-gram with backoff.* Paper presented at the International Conference on Natural Language Processing and Knowledge Engineering.

Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition* (2nd ed.): Pearson Education India.

Levenshtein, V. I. (1966). *Binary codes capable of correcting deletions, insertions and reversals.* Paper presented at the Soviet physics doklady.

Li, M., Zhang, Y., Zhu, M., & Zhou, M. (2006). *Exploring distributional similarity based models for query spelling correction.* Paper presented at the Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics.

Lounis, O., Guermeche, B., Eddine, S., Saoudi, L., & Benaicha, S. E. (2014). *A new algorithm for detecting SQL injection attack in Web application.* Paper presented at the Science and Information Conference (SAI), 2014.

Lu, W., Du, X., Hadjieleftheriou, M., & Ooi, B. C. (2014). Efficiently Supporting Edit Distance based String Similarity Search Using B-trees. *IEEE Transactions on Knowledge and Data Engineering, 26*(12), 2983-2996.

Lund, W. B. (2014). *Ensemble Methods for Historical Machine-Printed Document Recognition.* (PhD dissertation, Brigham Young University, Utah, United States).

Ma, D., & Agam, G. (2013). *A super resolution framework for low resolution document image OCR.* Paper presented at the IS&T/SPIE Electronic Imaging.

Maarif, H., Akmeliawati, R., Htike, Z., & Gunawan, T. S. (2014). *Complexity Algorithm Analysis for Edit Distance.* Paper presented at the International Conference on Computer and Communication Engineering (ICCCE), 2014

Mahdi, A. A. M. (2012). *Spell checking and correction for Arabic text recognition.* (Master thesis), King Fahd university of petroleum and minerals, Saudi Srabia.

Mainsah, B. O., Morton, K. D., Collins, L. M., Sellers, E. W., & Throckmorton, C. S. (2015). Moving away from error-related potentials to achieve spelling correction in P300 spellers. *IEEE Transactions on Neural Systems and Rehabilitation Engineering, 23*(5), 737-743.

Mihov, S., & Schulz, K. U. (2004). Fast approximate search in large dictionaries. *Computational Linguistics, 30*(4), 451-477.

Musa, S. M., Opara, E. U., Shayib, M. A., & Oliver, J. (2016). Measurement and Test Performance for Integrated Digital Loop Carrier for White Noise Impairment Using Interleaved Mode. *Communications of the IIMA, 14*(3), 4.

Naseem, T. (2004). *A Hybrid Approach for Urdu Spell Checking.* (Master thesis), National University of Computer & Emerging Sciences, Islamabad, Pakistan.

Naseem, T., & Hussain, S. (2007). A novel approach for ranking spelling error corrections for Urdu. *Language Resources and Evaluation, 41*(2), 117-128. doi: 10.1007/s10579-007-9028-6

Navarro, G. (2001). A Guided tour to approximate string matching. *ACM Computing Surveys (CSUR), 33*(1), 31-88.

Navarro, G., Grabowski, S., Mäkinen, V., & Deorowicz, S. (2005). Improved time and space complexities for transposition invariant string matching *Technical Report TR/DCC-2005-4, Department of Computer Science*: University of Chile.

Pal, S., & Rajasekaran, S. (2015). *Improved algorithms for finding edit distance based motifs.* Paper presented at the IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2015.

Phillips, C. R. (2015). *Employing an Efficient and Scalable Implementation of the Cost Sensitive Alternating Decision Tree Algorithm to Efficiently Link Person Records.* Master thesis, Texas State University, San Marcos, United States.

Polyanovsky, V., Roytberg, M. A., & Tumanyan, V. G. (2011). Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences. *Algorithms for Molecular Biology, 6*, 25.

Popescu, O., & Vo, N. P. A. (2014). *Fast and Accurate Misspelling Correction in Large Corpora.* Paper presented at the Conference on Empirical Methods in Natural Language Processing (EMNLP).

Pradhan, N., Gyanchandani, M., & Wadhvani, R. (2015). A Review on Text Similarity Technique used in IR and its Application. *International Journal of Computer Applications, 120*(9).

Rardin, R. L., & Uzsoy, R. (2001). Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics, 7*(3), 261-304.

Rieck, K., & Wressnegger, C. (2016). Harry: A Tool for Measuring String Similarity. *Journal of Machine Learning Research, 17*(9), 1-5.

Ruoro, S. W. (2009). *A Parallel Corpus Based Translation Using Sentence Similarity.* PhD dissertation, University of Nairobi, Nairobi, Kenya.

Sheng, C., Tao, Y., & Li, J. (2012). Exact and approximate algorithms for the most connected vertex problem. *ACM Transactions on Database Systems (TODS), 37*(2), 12.

Siklósi, B., Novák, A., & Prószéky, G. (2016). Context-aware correction of spelling errors in Hungarian medical documents. *Computer Speech & Language, 35,* 219-233.

Singh, S. P., Kumar, A., Darbari, H., Chauhan, S., Srivastava, N., & Singh, P. (2015). Evaluation of Similarity metrics for translation retrieval in the Hindi-English Translation Memory. *International Journal of Advanced Research in Computer and Communication Engineering, 4*(8).

Ukkonen, E. (1985). Finding approximate patterns in strings. *Journal of algorithms, 6*(1), 132-137.

Vargas, S. G. J. (2008). *A Knowledge-Based information Extraction Prototype for Data-Rich Documents in the Information Technology Domain.* (Master dissertation), National University of Colombia, Bogotá, Colombia.

Wick, M. L., Ross, M. G., & Learned-Miller, E. G. (2007). *Context-sensitive error correction: Using topic models to improve OCR.* Paper presented at the Ninth International Conference on Document Analysis and Recognition (ICDAR).

Appendix A

A Sample of Output Results of Word Length 3

Input		Output			
Word1	**Word2**	**Processing time (LA) in microsecond**	**Processing time (ILA-OT) in microsecond**	**Distance (LA)**	**Distance (ILA-OT)**
aba	baa	1.54	0.90	2	2
baa	cab	0.72	0.56	2	2
cab	bad	0.72	0.48	2	2
bad	dab	0.82	0.48	2	2
dab	bag	0.64	0.48	2	2
bag	gab	0.64	0.56	2	2
gab	bah	0.72	0.56	2	2
bah	jab	0.72	0.48	2	2
jab	kab	0.64	0.56	1	1
kab	alb	0.72	0.56	2	2
alb	bal	0.82	0.56	2	2
bal	lab	0.72	0.64	2	2
lab	bam	0.64	0.56	2	2
bam	ban	0.72	0.48	1	1
ban	nab	0.72	0.48	2	2
nab	abo	0.72	0.56	2	2
abo	boa	0.72	0.56	2	2
boa	bap	0.72	0.48	2	2
bap	arb	0.72	0.56	3	3
arb	bar	0.72	0.56	2	2
bar	bra	0.82	0.48	2	2

Appendix B

A Sample of Output Results of Word Length 4

Input		Output			
Word1	Word2	Processing time (LA) in microsecond	Processing time (ILA-OT) in microsecond	Distance (LA)	Distance (ILA-OT)
book	jock	1.14	0.82	2	2
jock	jivy	1.14	0.72	3	3
jivy	jamb	1.14	0.72	3	3
jamb	juku	1.14	0.72	3	3
juku	jack	1.14	0.72	3	3
jack	joky	1.14	0.72	3	3
joky	junk	1.14	0.72	3	3
junk	jiff	1.06	0.72	3	3
jiff	jimp	1.14	0.82	2	2
jimp	jauk	1.14	0.72	3	3
jauk	phiz	1.22	0.82	4	4
phiz	zouk	1.14	0.72	4	4
zouk	jibb	1.06	0.72	4	4
jibb	zonk	1.14	0.72	4	4
zonk	juke	1.14	0.72	4	4
juke	cozy	1.22	0.72	4	4
cozy	zyme	1.14	0.72	4	4
zyme	mazy	1.06	0.72	4	4
mazy	jouk	1.06	0.82	4	4
jouk	qoph	1.14	0.82	3	3
qoph	jink	1.06	0.72	4	4

Appendix C

A Sample of Output Results of Word Length 5

Input		Output			
Word1	**Word2**	**Processing time (LA) in microsecond**	**Processing time (ILA-OT) in microsecond**	**Distance (LA)**	**Distance (ILA-OT)**
aahed	abaca	1.96	1.06	4	4
abaca	abaci	1.64	1.06	1	1
abaci	aback	1.64	0.98	1	1
aback	abada	1.54	0.98	2	2
abada	abaft	1.54	1.06	2	2
abaft	aband	1.64	0.98	2	2
aband	abase	1.46	0.98	2	2
abase	abash	1.54	1.06	1	1
abash	abate	1.64	0.98	2	2
abate	abbes	1.64	0.98	3	3
abbes	abbey	1.72	0.98	1	1
abbey	abbot	1.54	0.98	2	2
abbot	abdal	1.54	0.98	3	3
abdal	abeam	1.64	1.06	2	2
abeam	abear	1.54	0.98	1	1
abear	abele	1.72	1.06	2	2
abele	aberr	1.46	0.98	2	2
aberr	abets	1.54	0.98	2	2
abets	abhal	1.64	1.06	3	3
abhal	abhor	1.54	0.98	2	2
abhor	abide	1.54	0.98	3	3

Appendix D

A Sample of Output Results of Word Length 6

Input		Output			
Word1	Word2	Processing time (LA) in microsecond	Processing time (ILA-OT) in microsecond	Distance (LA)	Distance (ILA-OT)
aahing	abacus	2.38	1.54	5	5
abacus	abaist	2.20	1.46	3	3
abaist	abanet	2.20	1.38	2	2
abanet	abanga	2.12	1.54	2	2
abanga	abased	2.12	1.38	3	3
abased	abaser	2.04	1.46	1	1
abaser	abases	2.04	1.46	1	1
abases	abasia	2.20	1.38	2	2
abasia	abassi	2.12	1.38	2	2
abassi	abated	2.12	1.38	3	3
abated	abater	2.12	1.46	1	1
abater	abates	2.28	1.46	1	1
abates	abatis	2.12	1.54	1	1
abatis	abator	2.12	1.46	2	2
abator	abawed	2.12	1.38	3	3
abawed	abbacy	2.38	1.54	4	4
abbacy	abbess	2.20	1.30	3	3
abbess	abbeys	2.12	1.46	1	1
abbeys	abbots	2.12	1.30	2	2
abbots	abbott	2.20	1.46	1	1
abbott	abbrev	2.04	1.38	3	3

Appendix E

A Sample of Output Results of Word Length 7

Input		Output			
Word1	Word2	Processing time (LA) in microsecond	Processing time (ILA-OT) in microsecond	Distance (LA)	Distance (ILA-OT)
aaronic	abacist	2.70	2.04	6	6
abacist	abactor	2.70	1.88	3	3
abactor	abaculi	2.62	1.80	3	3
abaculi	abaddon	2.62	1.96	4	4
abaddon	abaiser	2.62	1.72	4	4
abaiser	abalone	2.62	1.72	4	4
abalone	abandon	2.62	2.12	3	3
abandon	abandum	2.62	1.80	2	2
abandum	abasers	2.62	1.80	4	4
abasers	abashed	2.62	1.96	3	3
abashed	abashes	2.62	1.80	1	1
abashes	abasing	2.62	1.88	3	3
abasing	abassis	2.62	1.96	3	3
abassis	abaters	2.70	1.80	3	3
abaters	abating	2.62	1.88	3	3
abating	abattis	2.54	1.88	3	3
abattis	abature	2.62	1.72	3	3
abature	abaxial	2.62	1.80	4	4
abaxial	abaxile	2.62	1.96	2	2
abaxile	abbotcy	2.70	1.72	5	5
abbotcy	abdomen	2.62	1.80	4	4

Appendix F

A Sample of Output Results of Word Length 8

Input		Output			
Word1	**Word2**	**Processing time (LA) in microsecond**	**Processing time (ILA-OT) in microsecond**	**Distance (LA)**	**Distance (ILA-OT)**
children	aaaaaaaa	3.28	2.62	8	8
aaaaaaaa	aaaaaaah	3.28	2.28	1	1
aaaaaaah	aaaaahhh	3.36	2.28	2	2
aaaaahhh	aaaabbbb	3.52	2.38	4	4
aaaabbbb	aaaahhhh	3.28	2.28	4	4
aaaahhhh	aaahhhhh	3.52	2.38	1	1
aaahhhhh	aabbaabb	3.36	2.38	6	6
aabbaabb	aabbccdd	3.44	2.46	4	4
aabbccdd	aabbddrr	3.36	2.38	4	4
aabbddrr	aachener	3.28	2.20	5	5
aachener	aaddison	3.36	2.38	6	6
aaddison	aaheeric	3.28	2.28	6	6
aaheeric	aahesgit	3.36	2.46	3	3
aahesgit	aalaadin	3.36	2.20	5	5
aalaadin	aalenian	3.52	2.46	4	4
aalenian	aalesund	3.52	2.38	4	4
aalesund	aalsmeer	3.36	2.38	5	5
aalsmeer	aaltonen	3.44	2.46	4	4
aaltonen	aanderaa	3.44	2.38	6	6
aanderaa	aapianoc	3.44	2.28	6	6
aapianoc	aardvark	3.52	2.28	6	6

Appendix G

A Sample of Output Results of Word Length 9

Input		Output			
Word1	Word2	Processing time (LA) in microsecond	Processing time (ILA-OT) in microsecond	Distance (LA)	Distance (ILA-OT)
maximizer	aaaaaaaaa	4.26	3.02	9	9
aaaaaaaaa	aaaaaaaah	4.10	3.02	1	1
aaaaaaaah	aaaaahhhh	4.26	2.94	3	3
aaaaahhhh	aaaahhhhh	4.26	3.10	1	1
aaaahhhhh	aaabbbccc	4.26	2.94	6	6
aaabbbccc	aabbaccdd	4.42	2.94	4	4
aabbaccdd	aardvarks	4.42	3.02	7	7
aardvarks	aarestrup	4.26	3.02	5	5
aarestrup	aaronsohn	4.34	2.86	6	6
aaronsohn	abababab a	4.26	2.94	8	8
ababababa	ababbcbcc	4.42	3.02	4	4
ababbcbcc	ababcbcac	4.26	3.10	2	2
ababcbcac	abacadaea	4.34	2.94	5	5
abacadaea	abadinsky	4.18	2.86	6	6
abadinsky	abakaliki	4.42	3.02	5	5
abakaliki	abandoned	4.18	2.94	6	6
abandoned	abanooneo	4.34	2.94	2	2
abanooneo	abarbanel	4.18	3.02	4	4
abarbanel	abasement	4.26	2.94	6	6
abasement	abashidze	4.10	3.02	5	5
abashidze	abassides	4.26	3.02	3	3

Appendix H

A Sample of Output Results of Word Length 10

Input		Output			
Word1	**Word2**	**Processing time (LA) in microsecond**	**Processing time (ILA-OT) in microsecond**	**Distance (LA)**	**Distance (ILA-OT)**
complexify	aaaaaaaaaa	5.08	3.76	10	10
aaaaaaaaaa	aaaaabbbbb	5.00	3.76	5	5
aaaaabbbbb	aaaaahhhhh	5.08	3.76	5	5
aaaaahhhhh	aaaaattttt	5.00	3.60	5	5
aaaaattttt	aaaterials	4.92	3.68	7	7
aaaterials	aabbccddee	5.24	3.52	8	8
aabbccddee	aabboouutt	5.00	3.60	6	6
aabboouutt	aaeeanapoy	5.00	3.60	8	8
aaeeanapoy	aaehanapot	5.24	3.68	2	2
aaehanapot	aaehanapoy	4.92	3.68	1	1
aaehanapoy	aaesanapoy	5.16	3.52	1	1
aaesanapoy	aaezanapoy	5.00	3.76	1	1
aaezanapoy	aardenburg	5.08	3.76	7	7
aardenburg	aartolahti	5.00	3.68	7	7
aartolahti	ababababab	5.16	3.84	8	8
ababababab	abacterial	5.24	3.68	6	6
abacterial	abandoning	5.16	3.76	6	6
abandoning	abandonmen	5.08	3.68	3	3
abandonmen	abatements	5.16	3.76	6	6
abatements	abatzoglou	4.92	3.68	6	6
abatzoglou	abbaaccddc	5.08	3.68	8	8

Appendix I

A Sample of Output Results of Word Length 11

Input		Output			
Word1	Word2	Processing time (LA) in microsecond	Processing time (ILA-OT) in microsecond	Distance (LA)	Distance (ILA-OT)
environment	aaaaaaaaaaa	5.98	4.66	11	11
aaaaaaaaaaa	aaanagement	6.06	4.42	7	7
aaanagement	aaronovitch	6.22	4.50	9	9
aaronovitch	abakanowicz	6.06	4.42	6	6
abakanowicz	abandonment	6.14	4.58	8	8
abandonment	abbandonata	6.06	4.58	5	5
abbandonata	abbangement	6.22	4.58	6	6
abbangement	abberations	6.14	4.66	7	7
abberations	abbevillian	6.06	4.58	6	6
abbevillian	abbildungen	6.06	4.42	7	7
abbildungen	abbitbation	6.06	4.66	6	6
abbitbation	abbonamenti	6.14	4.58	8	8
abbonamenti	abbonamento	5.90	4.50	1	1
abbonamento	abbotsholme	6.14	4.50	7	7
abbotsholme	abbreviated	6.06	4.50	8	8
abbreviated	abbreviatio	5.90	4.76	2	2
abbreviatio	abcabcbacba	6.32	4.66	8	8
abcabcbacba	abcdefghijk	5.98	4.42	8	8
abcdefghijk	abcdefghjkl	5.82	4.42	2	2
abcdefghjkl	abcdehijklm	5.98	4.58	4	4
abcdehijklm	abdelrahman	6.14	4.58	8	8

Appendix J

A Sample of Output Results of Word Length 12

Input		Output			
Word1	**Word2**	**Processing time (LA) in microsecond**	**Processing time (ILA-OT) in microsecond**	**Distance (LA)**	**Distance (ILA-OT)**
adversatives	aaaaaaaaaaaa	6.96	5.40	12	12
aaaaaaaaaaaa	abababababab	6.96	5.58	6	6
abababababab	abandonments	7.14	5.40	9	9
abandonments	abbangements	7.30	5.40	4	4
abbangements	abbrevations	7.22	5.24	8	8
abbrevations	abbreviating	7.06	5.08	3	3
abbreviating	abbreviation	7.06	5.40	2	2
abbreviation	abbreviatons	6.88	5.58	2	2
abbreviatons	abbreviators	6.88	5.16	1	1
abbreviators	abcabcabcabc	7.38	5.50	10	10
abcabcabcabc	abcdabcdabcd	7.22	5.50	5	5
abcdabcdabcd	abcdefghijkl	7.14	5.32	8	8
abcdefghijkl	abcdefghiklm	6.80	5.32	2	2
abcdefghiklm	abchitectube	7.30	5.40	9	9
abchitectube	abchitecture	6.96	5.50	1	1
abchitecture	abderrahmane	7.30	5.40	9	9
abderrahmane	abencerrages	7.54	5.50	8	8
abencerrages	aberbrothick	7.30	5.40	9	9
aberbrothick	aberbrothock	6.96	5.24	1	1
aberbrothock	aberdonensis	7.06	5.32	8	8
aberdonensis	abhandlungen	7.22	5.40	9	9