PARALLEL QUICK-SKIP SEARCH HYBRID ALGORITHM FOR THE EXACT STRING MATCHING PROBLEM

By

MUSTAFA ABDULSAHIB NASER

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science

January 2010

DECLARATION

Name: ……...Mustafa Abdulsahib Naser ……..

Matric No: ……PCOM0066/08…….…….....

Faculty: …….....School of Computer Science ……..……..

Thesis Title: …….Parallel Quick-Skip Search Hybrid Algorithm for the Exact String Matching Problem

I hereby declare that this thesis in I have submitted to **School of Computer Science** on June **21th January 2010** is my own work. I have stated all references used for the completion of my thesis.

I agree to prepare electronic copies of the said thesis to the external examiner or internal examiner for the determination of amount of words used or to check on plagiarism should a request be made.

I make this declaration with the believe that what is stated in this declaration is true and the thesis as forwarded is free from plagiarism as provided under Rule 6 of the Universities and University Colleges (Amendment) Act 2008, University Science Malaysia Rules (Student Discipline) 1999.

I conscientiously believe and agree that the University can take disciplinary actions against me under Rule 48 of the Act should my thesis be found to be the work or ideas of other persons.

Students Signature: ..... Date: 21th January 2010

(Mustafa Abdulsahib Naser)

Acknowledgment of receipt by: Date:

بسم الله الرحمن الرحيم

فَأَمَّا الزَّبَدُ فَيَذْهَبُ جُفَاء وَأَمَّا مَا يَنفَعُ النَّاسَ فَيَمْكُثُ فِي الأَرْضِ كَذَلِكَ يَضْرِبُ اللّهُ الأَمْثَالَ

صدق الله العظيم

سورة الرعد اية١٧

DEDICATION

This thesis is dedicated to my wonderful parents, who have raised me to be the person I am today. You have been with me every step of the way, through good times and bad. Thank you for all the unconditional love, guidance, and support that you have always given me, helping me to succeed and instilling in me the confidence that I am capable of doing anything I put my mind to. Thank you for everything. I love you!

ACKNOWLEDGEMENTS

Firstly, All Praise is to Allah for giving me the courage and inspiration to finish this thesis. I am heartily thankful to my supervisor, Associate Professor Dr. Nur'Aini Abdul Rashid, whose encouragement, guidance and support from the initial to the final level that enabled me to develop an understanding of the subject in hand.

I owe my deepest gratitude to my friend and colleague Mohammed F. Eessa for his continuous encouragement and support, and to my friend Muhammed Zuhear Almulali for his guidance in correcting the language of this thesis.

I am indebted to my friends Bisam, Zaid, Talib, Mustafa, Hala and Khansaa for their support and encouragement. Without you this would not have happened.

This thesis would not have been possible unless for the support of Universiti Sains Malaysia (USM). I am grateful and much thankful.

Last but not least, my dearest family. Although you are not near but your unconditional love, support and prayers made me reach this important point in my life. Thank you.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

API	Application Program Interface
BF	Brute Force algorithm
BM	Boyer-Moore algorithm
BNDM	Backward Nondeterministic DAWG Matching algorithm
BR	Berry-Ravindran algorithm
CPU	Central Processing Unit
CRCW	Concurrent Read Concurrent Write
DAWG	Directed Acyclic Word Graph
DFA	Deterministic Finite Automaton
ED	Edit Distance
FDM	Forward DAWG Matching algorithm
FS	Fast Search algorithm
GB	Giga Byte
GH	Giga Hertz
KMP	Knuth-Morris-Pratt algorithm
KR	Karp Rabin algorithm
MB	Mega Byte
MCCRB	Mesh-Connected Computer with a Reconfigurable Bus system
MIMD	Multiple Instruction Multiple Data
MPI	Massage Passing Interface
NFA	Nondeterministic Finite Automata
OpenMP	Open Multi-Processing
POSIX	Portable Operating System Interface
PRAM	Parallel Random Access Machine

qsBc	Quick Search Bad Character
RAM	Random Access Machine
RF	Reverse Factor algorithm
SO	Shift-Or algorithm
SIMD	Single Instruction Multiple Data
VLDCs	Variable Length Don’t Cares

ALGORITMA HIBRID CARIAN CEPAT DAN LOMPAT YANG SELARI UNTUK MASALAH PADANAN RENTETAN YANG TEPAT

ABSTRAK

Masalah padanan rententan merupakan mercu tanda dalam kebanyakan bidang sains komputer kerana peranan yang dimainkannya dalam pelbagai aplikasi komputer. Oleh itu, beberapa algoritma padanan rententan telah dihasilkan dan diaplikasikan dalam kebanyakan sistem operasi, dapatan semula maklumat, penyunting, enjin carian internet, pintasan dinding-panas dan pola jujukan pencarian asid amino atau nukleotida dalam genom dan pangkalan data jujukan protein. Beberapa faktor penting dipertimbangkan semasa proses padanan, iaitu seperti jumlah perbandingan aksara, jumlah masa percubaan dan masa yang digunakan. Penyelidikan ini mencadangkan suatu algoritma padanan tali hibrid yang tepat dengan menggabungkan sifat terbaik daripada algoritma carian cepat dan carian lompat. Ini bertujuan mendemonstrasikan serta mereka kaedah yang lebih baik bagi menyelesaikan masalah padanan rententan pada kelajuan yang lebih tinggi dan kos yang lebih rendah. Hal ini dapat mempertingkatkan kehadiran serta prestasi urutan algoritma dengan menggunakan model yang berkongsi memori secara selari. Algoritma hibrid diuji menggunakan pelbagai jenis data piawai semasa fasa berjujukan dan selari. Algoritma hibrid juga memberikan keputusan yang lebih efisien dibandingkan dengan algoritma asal, dari segi bilangan aksara perbandingan dan bilangan percubaan apabila algoritma hibrid diaplikasikan secara berjujukan dengan panjang pola yang berbeza. Tambahan pula, keselarian algoritma hibrid telah meningkatkan kualiti prestasi metrik dengan kelajuan yang lebih tinggi, serta kecekapan dan peratusan yang lebih baik.

PARALLEL QUICK-SKIP SEARCH HYBRID ALGORITHM FOR THE EXACT STRING MATCHING PROBLEM

ABSTRACT

The string matching problem occupies a corner stone in many computer science fields because of the fundamental role it plays in various computer applications. Thus, several string matching algorithms have been produced and applied in most operating systems, information retrieval, editors, internet searching engines, firewall interception and searching nucleotide or amino acid sequence patterns in genome and protein sequence databases. Several important factors are considered during the matching process such as number of character comparisons, number of attempts and the consumed time. This research proposes a hybrid exact string matching algorithm by combining the good properties of the Quick Search and the Skip Search algorithms to demonstrate and devise a better method to solve the string matching problem with higher speed and lower cost by enhancing the existing algorithms sequentially and improving their performance using the parallel shared memory model. The hybrid algorithm was tested using different types of standard data during the sequential and parallel phases. The hybrid algorithm provides efficient results compared with the original algorithms in terms of number of character comparisons and number of attempts when the hybrid algorithm is sequentially applied with different pattern lengths. Additionally, parallelising the hybrid algorithm produced better quality in performance metrics through providing higher speedup with better efficiency and percentage of gain.

CHAPTER 1

INTRODUCTION

1.1 Introduction

This research concerns the string matching problem. A hybrid algorithm which solves the string matching problem and a parallel method for this hybrid algorithm will be proposed in this research. String matching is used to check the similarities of strings. To solve the string matching problem it is necessary to find an algorithm which can locate the similarities of strings. The string matching procedure is an algorithm which compares a short string called pattern with a long string called text, its function is to check whether this pattern is a substring of the text. The procedure outputs location when a pattern occurs in the text and produces a mismatched signal when no pattern occurs in the text. In many fields, such as computer science, computer engineering, bio-science, lexical analysis, database query and so on, string matching processing is essential and therefore applied frequently (Navarro and Raffinot, 2002).

There are different types of problems in string matching; the most fundamental problem is the exact string matching problem. The inputs of this problem are two strings as mentioned above, text and pattern, while the processes is to find all exact occurrences of the pattern string in the text string. This type of string matching algorithm comes in two types, namely the on-line and off-line algorithms. The on-line algorithms pre-process the pattern, but do not pre-process the text. On the other hand, the off-line algorithms pre-

process the text (Michailidis and Margaritis, 2000). This study focuses on on-line type of algorithms.

1.2 Definitions

A string matching problem can be defined as finding one or more occurrence of a given pattern string *P* of length *m* in a text string *T* of length *n*, which are built over a finite alphabet set Σ of size σ.

Definition 1.1: An alphabet Σ is a set of characters. The size of the alphabet is denoted by σ and represented by an integer number.

Definition 1.2: A string is a sequence of characters drawn from an alphabet. The input of the string matching algorithm are two strings, which are the pattern string $P = p_0\ p_1\ ...\ p_{m-1}$ and the text string $T = t_0\ t_1\ ...\ t_{n-1}$ where $n \geq m$. Different sizes of text string and different pattern string lengths will be used in this study.

Generally, string matching algorithms scan the text with the aid of the sliding window mechanism. This mechanism involves opening a window on the text of which its size is equal to the pattern length *m*. Then it is followed by a comparison between the characters of the window and the characters of the pattern. This specific work of character comparison is called an attempt. After matching or mismatching all of the pattern characters with the window characters, the window is shifted along the text according to the heuristics of each algorithm (Charras and Lecroq, 2004).

Definition 1.3: A shift is defined as a safe skip to the number of characters without missing any occurrence of the pattern in the text (Weinsberg et al., 2007).

Most of the on-line exact string matching algorithms pre-process the pattern before searching the text. The purpose of the pre-processing phase is to maximize the length of the shift during the searching phase and that happens by collecting information about the pattern before starting the search of the pattern in the text. The searching phase involves different approaches for scanning the text to find the pattern occurrences in the text (Lecroq, 1995).

The character comparison between the pattern and the text can be performed in different orders. Most of the string matching algorithms perform the comparisons from left to right like Knuth-Morris-Pratt (Michailidis and Margaritis, 2000), Karp Rabin (Karp and Rabin, 1987) and many other string matching algorithms. Another example of order is in the Boyer-More string matching algorithms (Boyer and Moore, 1977) and many other algorithms, where the comparison is performed from right to left. There are also string matching algorithms that may perform the character comparisons in any order or in some specific order. Horspool string matching algorithm performs comparison in any order while Two Way algorithm uses its own particular comparison order (Hassan, 2005).

1.3 Motivation

Development of the algorithms is considered a critical component in solving the problems when using the computer. The consumed time, performance, deficiency and cost are considered important factors in developing the algorithms. Many studies focus on the string matching problem. The hybrid algorithms are considered an example of such studies that deal with getting benefits from the original algorithms and overcome

their weaknesses. In addition, parallel algorithm shows how to solve a given problem faster by using multiple processors. However, there have been little empirical studies on hybrid with parallel algorithms. Quick Search and Skip Search string matching algorithms are considered in this study, and these algorithms differ in their technique, performance, efficiency and usage.

1.4 Problem Statement

The Quick Search and Skip Search string matching algorithms are good algorithms to find all the occurrences of the pattern in the text, however, both are associated with problems. The Quick Search is an efficient algorithm when using large alphabets with a short pattern during the text search (Lecroq, 1995; Klaib et al., 2007), but show less efficient behaviour for small alphabets with a long pattern. On the other hand, the work by (Charras et al., 1998) proposed an algorithm (Skip Search) that shows an efficient behaviour when using small alphabets with a long pattern.

Based on the reverse behaviour of the two existing algorithms which deals with different alphabet types and different pattern lengths, along with the long consumed time wasted in searching big sized data, the important question that needs to be answered is "*How to overcome the performance weaknesses of the two existing algorithms by proposing a hybrid algorithm which takes advantage of the positive characteristics of both algorithms to solve the string matching problem efficiently in any alphabet type and any pattern length during the sequential and the parallel phases?"*.

1.5 Objectives

The objectives of this research are:

1. To propose and implement a hybrid string matching algorithm based on Quick Search and Skip Search algorithms in order to improve searching results.
2. To parallelize the proposed algorithm in order to improve processing time.
3. To evaluate the sequential and parallel performance of the proposed algorithm.

1.6 Contributions

The expected contributions of this research are:

1. A hybrid string matching algorithm from combining Quick Search and Skip Search algorithms that gains the efficient advantages of their positive characteristics.
2. Improve the processing time and performance of the proposed hybrid string matching algorithm by using a parallelization method.

1.7 Justification

As mentioned in Section 1.4 of this chapter, the algorithms that were chosen in this study are Quick Search and Skip Search string matching algorithms to hybridize their good properties and to overcome their weaknesses. These two algorithms differ in technique, performance, efficiency and usage. The Quick Search algorithm is efficient when large alphabets with a short pattern are used while the Skip Search algorithm is efficient when small alphabets with a long pattern are used. According to these different

properties, this study aims to extract the advantages from each algorithm in a hybrid algorithm that is able to solve the string matching problem efficiently using any type of alphabet and any length of pattern. The hybrid algorithm will provide less number of character comparisons and attempts during the sequential phase and reducing the consumed time during the parallel phase.

1.8 Research Methodology

This section discusses the main points related to the methodology parts of this research. These parts include the research procedure, the theoretical framework and the research design.

1.8.1 Research Procedure

The procedure of this research comprises different steps, it began with collecting the data by downloading it from websites (Pizza and Chili Corpus, 2009). A standard benchmark type of data was used in this research which illustrates the common use to present the string matching applications. These types of data contain the DNA sequence, protein sequence and English text.

The second procedure step of this research consists of two phases. The first phase is designing a new hybrid algorithm from the original algorithms (Quick Search and Skip Search algorithms) through their analysis to extract the advantages of the positive characteristics of both algorithms. The second phase is the proposed parallelization method to improve the processing time of the proposed hybrid algorithm.

The final step is a comparison that consists of two phases. The first comparison phase occurs between the original algorithms and the proposed hybrid algorithm to show the positive properties of the proposed hybrid algorithm over the original algorithms. The second comparison phase occurs between the sequential and the parallel approach of the proposed hybrid algorithm to show the improvement gained in time between the two approaches. Figure 1.1 shows the steps of the procedure taken by this research.

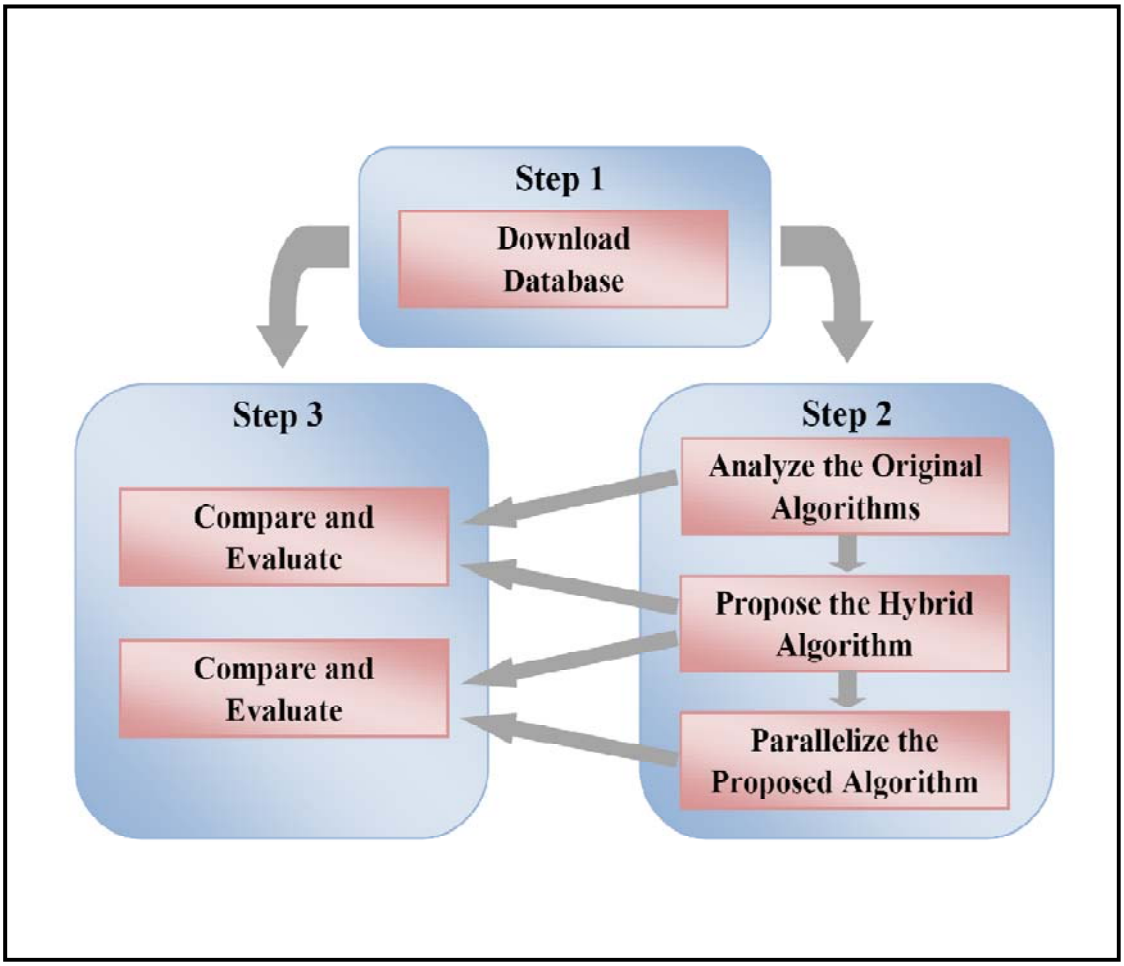


Figure 1.1: Research Procedure

1.8.2 Theoretical Framework

Several studies have addressed the string matching problem. These studies deal with producing new algorithms or hybridizing between the existing algorithms. The aim of the hybridization between the algorithms is mostly represented by creating a new efficient algorithm to solve the matching problem. Several examples of hybrid algorithms are discussed in Chapter 2 of this research. From this point the method of this research is generated which is represented by combining the good characteristics of the Quick Search algorithm (Sunday, 1990) and the Skip Search algorithm (Charras et al., 1998) in an algorithm which is capable of overcoming their weaknesses by solving the matching problem efficiently using different pattern lengths with different alphabet sizes.

1.8.3 Research Design

This section clarifies the attributes and variables involved in the research design. These attributes and variables include the purpose of the study, type of investigation, study setting and its time horizon. The purpose of the study will be a case study since the method is qualitative in nature. To prove the correctness of the study, the method is conducted by implementing steps to apply the hybrid algorithm. The type of investigation conducted will be “Causal” since the proposed algorithm is to add a new entity over the original algorithms. While the study’s setting is a “Lab Experiment” since it is presented by creating a new algorithm that will solve the matching problem by providing less number of character comparisons and less number of attempts during the searching operation. In addition, the implementation of the proposed method is a type of

lab environment. The time horizon for this research is a “Cross Sectional Study” since the data collected is necessary information that will help to design and implement other algorithms and there is no need to collect the resultant data in different situations and time.

1.9 Thesis Organization

This thesis is organized in five chapters. Chapter 1 begins by giving an introduction to the general fundamentals of string matching algorithms, and then it discusses the motivation and presents the problem statement of our study followed by the objective, contributions and justification of this thesis. The chapter ends with the brief explanation to the methodology part of this thesis. Chapter 2 offers a background and classification to the string matching algorithms, and also list and explains several previous studies of hybrid string matching algorithms. In addition, it will discuss some parallel issues and previous studies of parallelizing string matching algorithms. Chapter 3 outlines the method to the proposed hybrid algorithm in the sequential and parallel phases and also discusses the performance of each phase as well as the type of the data usage in the experimental results. Chapter 4 reports and evaluates the results of running the proposed hybrid algorithm during the sequential and parallel phases. Chapter 5 offers a conclusion of the major components of the research and suggests future work that may result in new features to be added to the proposed hybrid algorithm.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter presents a description of a few standard algorithms used in text processing. These algorithms are applied in various applications, for example, text editors, text compression, data retrieval systems, and computational biology. These algorithms are considered useful in different aspects and produce challenges in theoretical computer science. The literature survey distinguished two types of string matching solutions; the first one is when the pattern is fixed, while the second solution is when the text is fixed. This research focuses on the second type that is when the text is fixed and not known in progress.

Basically, there are two problems in string matching, the first one is the approximate string matching problem and the second is the exact string matching problem. This chapter briefly discusses the problem related to the approximate string matching explained in Section 2.2 for the purpose of understanding the basic idea behind it. Furthermore, the exact string matching problem is discussed by classifying it into different categories and provides examples and comparisons of the algorithms for each category as explained in Section 2.3. In addition, examples of the well known hybrid string matching algorithms are discussed in Section 2.4. Also, Section 2.5 will discuss some parallelism issues and will refer to some examples of parallelization techniques for string matching algorithms. Finally Section 2.6 concludes this chapter.

2.2 Approximate String Matching

The approximate string matching problem is defined as follows: a pattern string $P = p_0\, p_1 \ldots\, p_{m-1}$ and a text string $T = t_0\, t_1 \ldots\, t_{n-1},\ n \geq m,$ are given and a maximal number *K* of errors allowed between the pattern and its occurrences in the text (Navarro and Raffinot, 2002).

The solution of the approximate string matching algorithms can be off-line when the text can be pre-processed. It can also be on-line when the text is not known in advance. This consists of two phases, the pre-processing phase for the pattern and the searching phase of the pattern in the text. The pre-processing phase consists of an assembly of information about the pattern, while the searching phase which involves different approaches consisted of scanning the text to find all approximate occurrences of the pattern in the text. Some of these approaches include dynamic programming, deterministic finite automata, bit-parallelism and filtering algorithms approach. There are many different models in the approximate string matching problem with *K* number of errors. One of the best studied cases of this error model is the edit distance (ED) or Levenshtein's distance. It supports three main operations which are insertion, deletion and substitution of simple characters to edit the difference between two strings. As a definition, the edit distance is the minimum number of edit operation between two strings *X* and *Y* (ED(*X, Y*)), which is required to convert *X* to *Y*, or vice versa (Navarro et al., 2001). The following is a simple example for each edit distance operation.

2.2.1 Insertion

A character *F* of *X* is missing in *Y* at a corresponding position. By inserting a character *F* in this missing position, the string *Y* can be transformed into string *X* as shown in Figure 2.1.

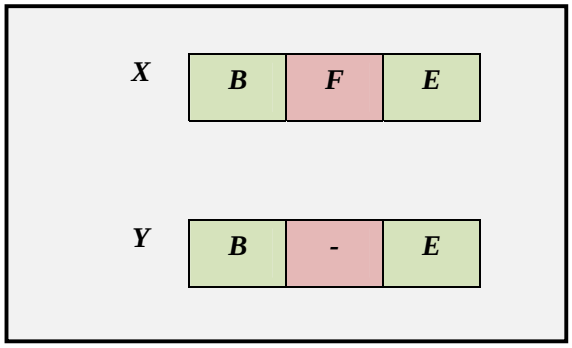


Figure 2.1: Insertion Operation

2.2.2 Substitutions

The symbols at corresponding positions are different. By substituting a character *D* to *C* in string *Y*, the string *Y* can be transformed into string *X* as shown in Figure 2.2.

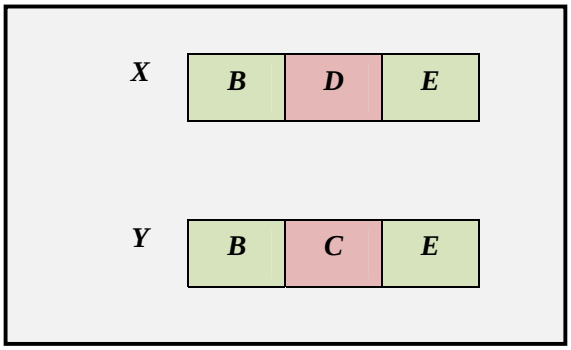


Figure 2.2: Substitution Operation

2.2.3 Deletion

A character *C* of *Y* is missing in *X* at a corresponding position as shown in Figure 2.3. By deleting a character *C* in string *Y*, the string *Y* can be transformed into string *X*.

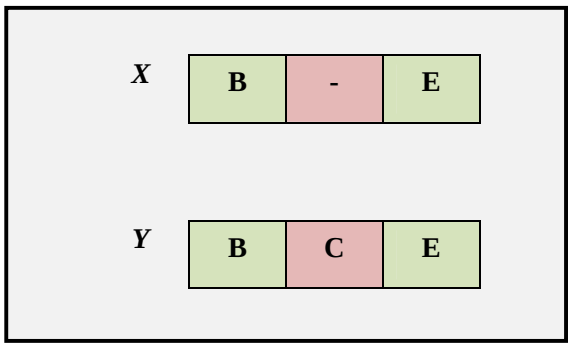
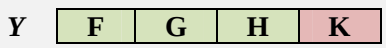
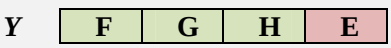


Figure 2.3: Deletion Operation

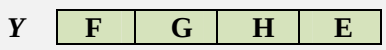
Example of the edit distance operations can be shown in Figure 2.4

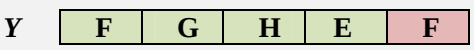
Given a string *X* =" *FGHEF* " and a string *Y* =" *FGHK*" . We want to transform *Y* into *X*. Finally, the ED(*X*, *Y*) = 2. First, substitute a character *E* to *K*.



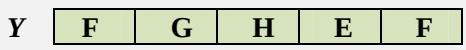


Insert a character *F* at the end of string *Y*.





Y successfully transforms to *X*.



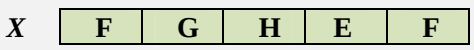


Figure 2.4: Example of Edit Distance Operation

In an approximate string matching problem, the number of *K* must be less than the length of the pattern $0 < K < m$, otherwise every text substring of length *m* will be converted into *P* by substituting the *m* characters. Therefore, the exact string problem is a special case of the approximate string matching problem in which *K*=0 (Michailidis and Margaritis, 2002).

2.3 Exact String Matching

The exact string matching problem is defined as follows: a text $T = t_0\, t_1 \ldots t_{n-1}$, and a pattern $P = p_0\, p_1 \ldots p_{m-1}$, where $n \geq m$. The purpose is to find all occurrences of *P* in *T*. All the characters in text *T* and pattern *P* are obtained from a finite set of characters called an alphabet denoted by Σ of size σ. The Figure 2.5 below shows a simple example of the exact string matching operation (Navarro and Raffinot, 2002).

Input: We are given a text *T* with length *n* and a pattern *P* with length *m*.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T **=**	**A**	**T**	**C**	**T**	**A**	**C**	**G**	**A**	**T**	**A**	**T**	**A**	**C**	**G**
P **=**	**A**	**C**	**G**											

n **=14,** ***m*** **= 3,** **Σ = (A, C, G, T),** ***σ*** **= 4**

Output: All the occurrences of pattern *P* in text *T*.

P occurs at location 5 and 12 of *T*

Figure 2.5: An Example of Exact String Matching Operation

Depending on whether the text or the pattern needs pre-processing, the exact string matching algorithms can be divided into two types. The first one is off-line exact string matching algorithms in which the text can be pre-processed. The second type is on-line exact string matching algorithms in which the text is not known in progress. This research deals with the on-line type of algorithms which is consisting of two phases, the pre-processing phase of the pattern and the searching phase for the pattern in the text. This section uses the same classifications used by (Michailidis and Margaritis, 2000). The authors classify the exact string matching algorithms into four main categories of algorithms as a survey to introduce several examples of exact string matching algorithms. The four categories are Classical algorithms, Suffix Automata algorithms, Bit-Parallelism algorithms and Hashing algorithms as shown in Figure 2.6.

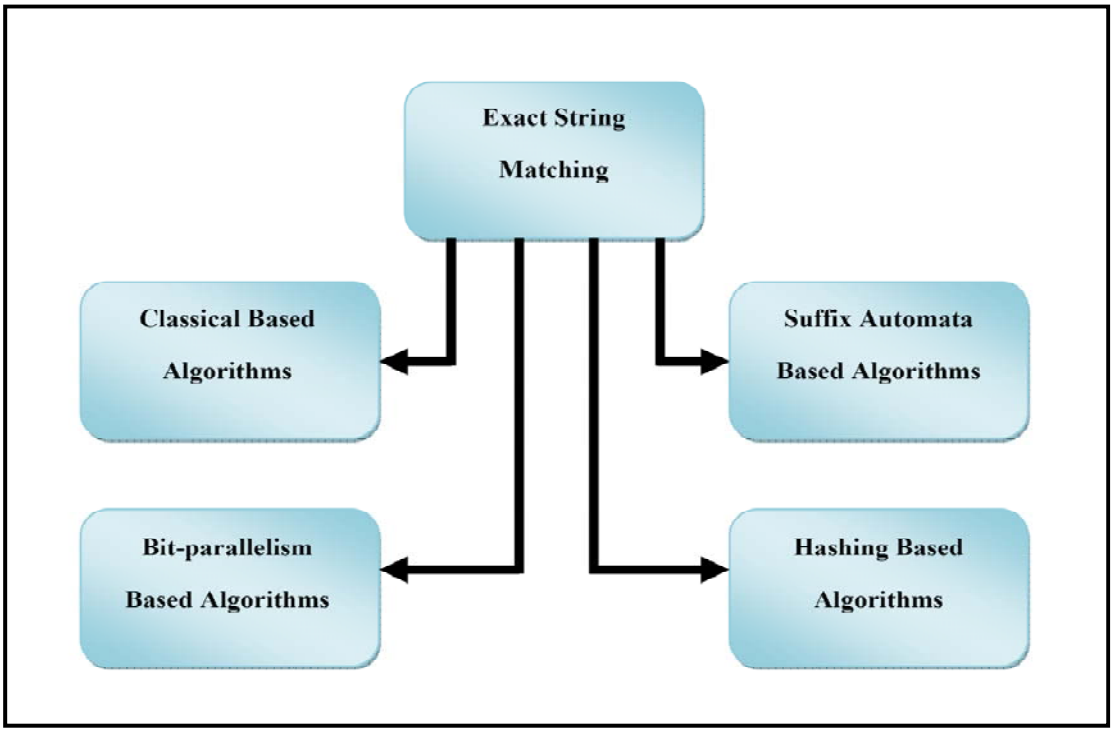


Figure 2.6: Exact String Matching Categories

2.3.1 Classical Based Algorithms

This category of string matching algorithms is based on character comparisons between a character in the text and a character in the pattern from different directions depending on the heuristic of the algorithm. The following are examples of some classical algorithms.

(A) Brute Force Algorithm

The Brute Force algorithm (BF) is a basic algorithm for the string matching problem. It locates all occurrences of pattern string in text string in time O(*mn*), where *n* is the length of the text and *m* is the length of the pattern. The algorithm finds all *n*-*m*+1 possible substrings with length *m* in the text by shifting a sliding window with length *m* to find all occurrences of the pattern in the text. The algorithm performs the comparison from left to right one by one and requires no pre-processing phase or extra space.

For every possible substring in the text, the Brute Force algorithm takes O(*m*) time to find whether the pattern appears in the text. Thus, the time complexity is O(*nm*). This algorithm is not efficient, because once a mismatch occurs, the sliding window shifts one position to the left and restarts to match from the first position of the pattern (Charras and Lecroq, 2004).

(B) Search with an Automata Algorithm

The main concept of this algorithm depends on the automaton theory by building the minimal Deterministic Finite Automaton (DFA) to the pattern as a pre-processing

phase. It is applied afterwards to search for the pattern in the text by parsing the text with the Deterministic Finite Automaton (Nedjah, 1998).

The search with an automata algorithm requires O($m\sigma$) time and space complexity to construct the minimal Deterministic Finite Automaton for the pre-processing phase, while it requires O(n) time complexity for the searching phase (Rafiq et al., 2004).

(C) Knuth-Morris-Pratt Algorithm

Discovered in 1977, the Knuth-Morris-Pratt (KMP) is the first linear time string matching algorithm which performs character comparisons from left to right in the pattern. When a mismatch occurs, the Knuth-Morris-Pratt algorithm moves the pattern to the right by maintaining the longest overlap of a prefix of the pattern with a suffix of the part of the text that has matched the pattern. This means that the algorithm uses the knowledge of the previous characters that were already examined in order to compute the next position of the pattern to use (Michailidis and Margaritis, 2000).

The Knuth-Morris-Pratt algorithm require O(m) time and space complexity for the pre-processing phase, while it requires O(n+m) time complexity for the searching phase (Charras and Lecroq, 2004).

(D) Boyer-Moore Algorithm

Discovered in 1977, the Boyer-Moore (BM) algorithm is considered as the most efficient string matching algorithm in usual applications. Its main function is to perform character comparisons from right to left in the pattern. The Boyer-Moore algorithm

triggered two heuristics which are bad character and good suffix on a mismatch to reduce the number of comparisons. The heuristics are independent and they are used simultaneously where the maximum shift computed by the two heuristics is considered after each attempt during the searching phase (Boyer and Moore, 1977).

There are many variants of Boyer-Moore algorithm, some of these variants like Galil Giancarlo algorithm, Apostolico Giancarlo algorithm and Turbo-Boyer-Moore algorithm use the same two heuristics that are used in the original Boyer-Moore algorithm and also perform the character comparisons from right to left with some modification. All those modifications generally aim at reducing the implementation complexity, space requirements, and search time. Another variant like Tuned Boyer-Moore algorithm and Horspool algorithm depend on simplifying the original Boyer-Moore algorithm by using only the bad character heuristic (Lecroq, 1995).

The Quick Search algorithm is another variant that simplifies the Boyer-Moore algorithm by using only the bad character heuristic, but it performs the character comparisons from left to right during the matching process (Sunday, 1990).

The Boyer-Moore algorithm requires $O(m+\sigma)$ time and space complexity for the heuristics' pre-processing phase, while it requires $O(nm)$ time complexity for the searching phase (Charras and Lecroq, 2004).

(E) Skip Search Algorithm

The idea behind the Skip Search algorithm is to build buckets as a pre-processing phase. These buckets contain information about all the alphabets' position in the pattern.

These alphabets start recording from the first left position of the pattern. The algorithm examines the *m*-th text character to delimit a possible starting search point.

There are two variants to the original Skip Search algorithm. The first variant is linear in time which combines the Skip Search algorithm with Knuth-Morris-Pratt algorithm; this algorithm is named the KMP Skip Search algorithm. The second variant is Alpha Skip Search algorithm. It improves the Skip Search algorithm by building a trie of length ($\log_\sigma m$) for all factors of the pattern, where there is a single bucket for each leaf of the trie (Charras et al., 1998).

The Skip Search requires O(m+σ) time and space complexity for the pre-processing phase, while it requires O(mn) time complexity for the searching phase (Rafiq et al., 2004).

2.3.2 Suffix Automata Based Algorithms

This category of string matching algorithms is based on using the suffix automaton data structure which is also called DAWG (Directed Acyclic Word Graph). Basically, this structure recognizes all the suffixes of the pattern (Michailidis and Margaritis, 2000). The following algorithms are types of this category.

(A) Reverse Factor Algorithm

The Reverse Factor (RF) algorithm scans the characters of the text from right to left using the smallest suffix automaton of the reverse pattern. It is even better than the Boyer-Moore family, because the shifts are longer.

The pre-processing phase of the Reverse Factor algorithm consists of computing the smallest suffix automaton for the reverse pattern, denoted as P^R. In the search phase, the algorithm parses the characters of the window from right to left with the automaton of the reverse pattern starting with the initial state. It continues until there are no more transitions defined for the current character of the window from the current state of the automaton. Then a right shift will be performed based on the length of the longest prefix of the reverse pattern P^R that has been matched (Crochemore et al., 1992).

The Reverse Factor algorithm requires O(*m*) time and space complexity for the pre-processing phase, while it requires O(*n log m /m*) time complexity for the searching phase (Charras and Lecroq, 2004).

(B) Forward DAWG Matching Algorithm

The Forward DAWG Matching (FDM) algorithm computes the smallest suffix automaton of the pattern as a pre-processing phase to obtain the longest factor of this pattern which ends at each position in the text. The searching phase performs the comparisons from left to right by parsing the text character with the suffix automaton (Charras and Lecroq, 2004).

2.3.3 Bit-parallelism Based Algorithms

The main function of the Bit-parallel algorithms is that they store several data items into a single computer word and then update them in parallel using a single computer operation. The technique of using bit-parallelism is a general way to simulate

simple nondeterministic finite automata (NFA) structure with simplicity, flexibility and no buffering as advantages of this approach (Prasad and Agarwal, 2008).

(A) Shift-Or Algorithm

The Shift-Or (SO) algorithm represent the state of the search as a number and each search iteration costs a small number of arithmetic and logical operations. The implementation of the algorithm is considered very competitive if the length of the pattern is smaller than the size of the computer word. The pre-processing phase consists of creating a table which has one bit mask for every character in the alphabet, and this bit mask corresponds to the position of the character in the pattern. The Shift-Or algorithm performs character comparisons from left to right in the pattern and involves keeping a set of all the prefixes of the pattern that match a suffix of the text (Baeza-Yates and Gonnet, 1992).

The Shift-Or algorithm requires $O(m+\sigma)$ time and space complexity for the pre-processing phase, while it requires $O(n)$ time complexity for the searching phase (Charras and Lecroq, 2004).

(B) Backward Nondeterministic DAWG Matching Algorithm

The Backward Nondeterministic DAWG Matching (BNDM) algorithm is considered a variant of the reverse factor algorithm. It therefore uses the suffix automaton of the reverse pattern but in a nondeterministic form which is simulated using bit-parallelism. Like Shift-OR algorithm, BNDM algorithm uses a table which has one

bit mask for every character in the alphabet and it is very efficient if the length of the pattern is smaller than the size of the computer word (Navarro and Raffinot, 1998).

The Backward Nondeterministic DAWG Matching algorithm requires O($m+\sigma$) time and space complexity for the pre-processing phase, while it requires O(nm) in the worse case and O($n \log m / m$) in the average case time complexity for the searching phase (Rafiq et al., 2004).

2.3.4 Hashing Based Algorithms

The main function of the hashing algorithms is to check out the similarity of the window content and the pattern, instead of checking at each text position if the pattern occurs. This approach provides a simple and efficient method of avoiding quadratic number of character comparisons in most practical situations (Lecroq, 2007).

(A) Karp Rabin Algorithm

The Karp Rabin (KR) algorithm searches for a pattern in a text by hashing. The algorithm checks the similarity between the pattern and the window in the text by computing the hashing function (H). This function is efficiently computable and highly recognized for strings.

The pre-processing phase of the Karp Rabin algorithm consists of computing the hash function for the pattern ($H(P)$). During the searching phase, the algorithm compares the ($H(P)$) with ($H(T[j...j+m-1])$) for $0 \leq j < n - m$ and if equality is found the algorithm checks the equality of $P = T[j...j+m-1]$ character by character (Karp and Rabin, 1987).

The Karp Rabin algorithm requires O(m) time and space complexity for the pre-processing phase, while it requires O(nm) time complexity for the searching phase and O($n+m$) time complexity for the expected number of text character comparisons (Charras and Lecroq, 2004).

The comparison of different foregoing linear exact string matching algorithms can be shown in Table 2.1 in terms of the pre-processing phase and the searching phase's time and space complexity with the important characteristics for each algorithm. The complexity information is obtained from the publications by (Charras and Lecroq, 2004; Rafiq et al., 2004), while the underlying technique is obtained from the original publication of the algorithms. These algorithms use different techniques in the pre-processing and searching phase. Furthermore, most of these algorithms pre-process the pattern before searching for the pattern in the text in order to maximize the length of the shifts during the searching phase. However this pre-processing phase needs extra space linear to the length of the pattern.

Brute Force (BF), the first mentioned algorithm, scans the character of the window from left to right and shifts the window exactly one position to the right after a mismatch or a complete match. The Knuth-Morris-Pratt (KMP) algorithm is an improvement of the Brute Force (BF) algorithm, which uses a shift function based on the notion of the prefixes of the pattern and it is considered the first linear string matching algorithm. Skip Search and KMP Skip Search algorithms behave like Knuth-Morris-Pratt algorithm by performing the characters of the window from left to right while the algorithms use buckets to determine the starting positions of the window in the text. The Boyer-Moore (BM) algorithm is considered as one of the most efficient string matching algorithms

which scan the characters of the window from right to left. There are many variants of Boyer-Moore algorithm which are widely recognized and used in various string matching applications. The work of many algorithms depend on automaton theory with the Knuth-Morris-Pratt or Boyer-Moore algorithms concepts. Search with an Automaton algorithm and Forward DAWG Matching (FDM) algorithm work with the concept of the Knuth-Morris-Pratt (KMP) algorithm by performing the character comparisons from left to right. Search with an Automaton algorithm use the minimal Deterministic Finite Automaton (DFA), while Forward DAWG Matching algorithm uses the suffix automaton. Like the Boyer-Moore type algorithms, the Reverse Factor algorithm scans the characters of the window from right to left by calculating the smallest suffix automaton in the deterministic form of the reverse pattern. Some of the algorithms use the nondeterministic form of the automata. Backward Nondeterministic DAWG Matching (BNDM) algorithm uses the suffix automaton of the reverse pattern in nondeterministic form which is simulated by using bit-parallelism. Shift-Or (SO) algorithm uses bit-wise operations for its work and Karp Rabin (KR) algorithm uses the hashing methodology for string searching.