



The Libyan Academy

School of Engineering and Applied Science

Department of Medical Engineering

Division of Genetic Engineering

BioQt an Integrated Bioinformatics Software Development *Kit*

**A Thesis Submitted to the Department of Medical Engineering in Partial
Fulfillment of the Requirements for Master Science Degree in Genetic
Engineering.**

By: Usama Shihub Erawab *Under Supervision of: Dr. Saeed Zamit*

Department Of Medical Engineering

Spring 2015

ABSTRACT:

Bioinformatics is a multi-disciplinary science focusing on the applications of computational methods and mathematical statistics to molecular biology. Choosing bioinformatics as specialization gives an opportunity to get involved with the most interesting computational techniques dealing with biological data to contribute to cure and diagnose some of genetic disorders that affect biological machines. The purpose of this library (which defines namespace [BioQt](#)), is to provide a set of routines for handling biological sequence data for Qt/C++ users (the full source code available on <https://github.com/alrawab/BioQt>). This thesis will shed the light on some modules of BioQt SDK such as exact string matching problem, Microsatellite Repeats, Palindromic sequences and sequence alignment algorithms (Longest Common Subsequence, Needleman-Wunsch and Smith-Waterman). This thesis examines and evaluates these challenging problems in bioinformatics by using Qt/C++.

Contents

1- Chapter One	1
Introduction	1
1.1 Bioinformatics:.....	1
1.2 The importance of bioinformatics:.....	2
1.3 Development of bioinformatics:.....	2
1.4 Bioinformatics application areas:.....	3
1.4.1 Sequence analysis:	3
1.4.2 Genome annotation:.....	4
1.4.3 Drug production:.....	4
1.4.4 Mutations detection:.....	4
1.4.5 Phylogeny:	5
1.4.6 Protein expression analysis:.....	5
1.4.7 Gene expression analysis:.....	5
1.4.8 Protein structure prediction:.....	5
1.4.9 Primer design:	6
1.5 Bioinformatics centers:.....	6
1.6 Qt:.....	7
1.7 Nucleic Acids:	7
1.8 Proteins:.....	9
1.9 Motivations:.....	9
1.10 Objective:	10
2 - Chapter Two:	11
String Processing Problems	11
2.1 Keywords:.....	11

2.2	Basic Definitions on String:	12
2.2.1	Alphabet:.....	12
2.2.2	Sequence (or string):	12
2.2.3	Subsequence:.....	12
2.2.4	Substring:	12
2.2.5	Length of string:.....	13
2.2.6	Prefix:.....	13
2.2.7	Proper prefix:	13
2.2.8	Suffix:.....	13
2.2.9	Proper Suffix:.....	13
2.3	Microsatellite Repeats:	14
2.4	Objective:	14
2.5	Trinucleotide repeats disorder:.....	15
2.5.1	Polyglutamine Diseases:	15
	Table 2.1 Polyglutamine (PolyQ) Diseases	16
2.5.2	Non-Polyglutamine Diseases:.....	16
	Table 2.2 Non-Polyglutamine Diseases.....	17
2.6	Material:	17
2.7	Methods:	18
2.8	Palindromes:	19
2.9	Importance of Palindromic sequence:	19
2.10	Objective:	20
2.11	Material:	20
2.12	Methods:.....	20
2.13	Exact String Matching:	23
2.14	Objective:	23
2.15	Material:	23

2.16	Methods:.....	24
2.17	Brute Force:.....	24
2.17.1	Brute Force Visualization:	25
2.18	The Knuth-Morris-Pratt (KMP) Algorithm:	26
2.18.1	The prefix-function π :	26
2.18.2	To compute π for the pattern 'p':.....	29
2.18.3	Find matches algorithm:	30
2.18.4	Advantage:	38
2.18.5	Disadvantages:	38
2.19	Boyer-Moore Algorithm:	38
2.19.1	Analysis of Boyer-Moore Algorithm:	39
2.19.2	Bad character Heuristic:.....	40
2.19.3	Good suffix Heuristic:.....	40
2.19.4	Advantages:.....	43
2.19.5	Disadvantages:	43
3	- Chapter Three:	44
	String Matching Evaluation Methods	44
3.1	Sequence Alignment:	45
3.2	Biological Motivation.....	45
3.3	Terminology:	46
3.3.1	Identity:	46
3.3.2	Similarity:	46
3.3.3	Homology:	46
3.4	Pairwise Alignment:	46
3.5	Multiple Sequence Alignment:.....	48
3.6	Pairwise Sequence Alignment by Dynamic Programming:.....	48
3.7	Global Alignment (Needleman-Wunsch Algorithm):.....	49

3.8	Local Alignment (Smith-Waterman Algorithm):.....	52
3.9	Substitution matrices:	54
3.9.1	Different kind Of Matrices:	55
3.9.2	Limitations:	56
3.10	Objective:	56
3.11	Material:	56
3.12	Methods:.....	56
-	Dynamic programing approach:.....	56
➤	Global alignment (Needleman-Wunsch algorithm).....	56
➤	Local alignment (Smith-Waterman algorithm).....	56
4	- Chapter Four	57
	Case Study and Results	57
4.1	Microsatellite Repeats Case Study:.....	57
	Motif	57
	Motif	58
4.2	Palindromes Case Study:.....	61
4.2.1	Results:.....	61
4.3	Exact String Matching:.....	64
4.3.1	Experiment:.....	64
4.3.2	Results:.....	64
4.4	Sequence Alignment:	68
4.4.1	Experiment:.....	68
4.4.2	Results:.....	68
5	- Chapter Five:	73
	Conclusion and Future work	73
6	- Refrences:	74

In God we trust

1- Chapter One

Introduction

This is an introductory chapter introduce the fundamental concepts about bioinformatics (Sections 1.1 to 1.5), Qt framework Section(1.6), biological macromolecules Sections(1.7 to 1.8) and Finally in both Sections 1.9 and 1.10 describes the motivations and the objectives of this thesis respectively.

1.1 Bioinformatics:

May not be a clear-cut definition and measurement of bioinformatics as term but it can be referred to it as “the administration, dealing and exploitation of biological information by using computer methods” that is why this field is one of overlapping fields of knowledge among other sciences. It is clear that this discipline is the merger between the various branches of life sciences and computer science but for accuracy this field guided for molecular science in particular and so can be called the Computational Molecular Biology which includes the Biosimulation , Bioimaging and others to pour in the end to the management and analysis of biological information. Hence it is clear that bioinformatics coming from the analysis of biomolecules such as DNA, RNA and Proteins on the one hand and on the other hand the computer classifying and arranging of this information, which is termed as Data Biomining.

1.2 The importance of bioinformatics:

No room for doubt that the tremendous progress and unprecedented in the field of bioinformatics which resulted in a novel and amazing scientific discoveries was a result of massively generated data produced by genomic research's in the past two decade. This has led to increased interest in them at all aspects. It has entered into under -graduate and graduate curricula and areas of scientific research on different backgrounds. In field of bioinformatics and studies that conducted in silico (literally Latin for "in silicon", alluding to use of silicon for semiconductor computer chips which mean the performance committed via computers) plans are designed in ideal conditions by building computer models of living cells which include the internal metabolic path-ways based on wet-lab-data (any materials are handled in liquid or volatile state) and applies them against in-silico simulation to reach the best possible theoretical results before they are applied to the real production of drugs , vaccines, hormones or any other biological compounds .As result of what mentioned above it's become crucial for emerge of new standalone discipline deals effectively with various data types (DNA , protein sequencing and coding regions.. etc.) resulted from those experiments and exploited optimally to increase the understand the living cells in terms of function and structure .

1.3 Development of bioinformatics:

The term bioinformatics has been formally used recently but its actual start was in the sixties of the last century by *Margaret O. Dayhoff* [1] the biochemist researcher at the National Biomedical Research Foundation (<http://www.nabr.org/>). Where she pioneered the application of mathematics and computational methods to the field of biochemistry and issued the first database of protein sequences in a series

of releases lasted until the seventies of the last century. In 1970 Needleman and Wunsch [2] introduce an algorithm that searches the similarities in the amino acid sequence of two proteins. Since the eighties of the last century has witnessed a steady growth in the number of genomes that have been identified there sequences. From the foregoing it is clear that bioinformatics beyond the bimolecular aspects also needs:

1. An effective data warehouse.
2. Availability of data which stored in data warehouse.
3. Manipulation of stored data by different methods to extract useful information.

1.4 Bioinformatics application areas:

In 1979 Paulien Hogeweg [3] coined the term bioinformatics for the study of informatics processes in biotic systems and since that this area has become expanding day after day. Some of the grand area of research in bioinformatics includes:

1.4.1 Sequence analysis:

It's the most elementary procedure in bioinformatics .This procedure focusing on the similarity and dissimilarity portions of biological sequence during medical analysis and genome mapping processes and implies subjecting a DNA or peptide sequence to sequence alignment, repeated sequence searches, or other bioinformatics methods.

1.4.2 Genome annotation:

In the context of genomics annotation is the process of identifying the locations of genes and determination of their features in a DNA sequence.

1.4.3 Drug production:

By applying the in silico subtractive genomics approach in the designing of effective drugs by subtraction of sequence between the host and parasite proteome provides information for a set of proteins that are likely to be essential to the parasite but absent in the host.

1.4.4 Mutations detection:

Traditionally genetic disorders are grouped into three cardinal categories: single-gene, chromosomal, and multifactorial disorders. Molecular and cytogenetic techniques have been applied to identify genetic mutations leading to diseases. Accurate diagnosis of diseases is essential for appropriate treatment of patients, genetic counseling and prevention strategies. To manage the sheer volume of sequence data produced. The bioinformatics create new algorithms and software to compare the sequencing results to the growing collection of human genome sequences and gremlin polymorphisms. New physical detection technologies are employed, such as oligonucleotide microarrays to identify chromosomal gains and losses and single-nucleotide polymorphism arrays to detect known point mutations.

1.4.5 Phylogeny:

It's important to understand evolutionary and genetic relationships between organisms; bioinformatics is very helpful to find out the dimensional time for evolution through molecular sequencing data and morphological data matrices.

1.4.6 Protein expression analysis:

Gene expression is measured in many ways including *mRNA* and protein expression however protein expression is one of the best clues of actual gene activity since proteins are usually final catalysts of cell activity. *Protein, microarrays* and high throughput (*HT*) mass spectrometry (*MS*) can provide a snapshot of the proteins present in a biological sample. Bioinformatics is very much involved in making sense of protein microarray and HT MS data.

1.4.7 Gene expression analysis:

The expression of many genes can be determined by measuring *mRNA* levels with various techniques such as microarrays, expressed *cDNA* sequence tag (*EST*) sequencing, serial analysis of gene expression (*SAGE*) tag sequencing, massively parallel signature sequencing (*MPSS*), or various applications of multiplexed in-situ hybridization etc. All of these techniques are extremely noise-prone and subject to bias in the biological measurement.

1.4.8 Protein structure prediction:

Polypeptide chain (so-called primary structure) can be easily determined from the sequence on the gene that codes for it and this primary structure uniquely determines a structure in its native environment. To understand the function for a protein it's vital to have

proper knowledge of this structure. For lack of better terms the protein structure is classified as secondary, tertiary and quaternary structure. Prediction of protein structure is very important task for bioinformatics for drug design and the design of novel enzymes.

1.4.9 Primer design:

In the past decade molecular biology has been transformed from the art of cloning a single gene to a statistical science measuring and calculating properties of entire genomes. It's become an urgent necessity for effective Programs for designing appropriate primers for polymerase chain reaction (PCR).

1.5 Bioinformatics centers:

There are many bioinformatics centers distributed worldwide providing services and resources for researchers and they available online. These centers vary in terms of comprehensiveness and capability and they can be classified in terms of importance as follows:

- **NCBI** (The National Center for Biotechnology Information): U.S. government-funded national resource for molecular biology information and exhibits a huge amount of diverse data Access to Many public databases and other references.
- **EBI** (The European Bioinformatics Institute).
- **EMBL** (The European Molecular Biology Laboratory).
- **DDBJ** (DNA database of Japan).

1.6 Qt:

Qt (“cute”) is a cross-platform development framework that can be run on various software and hardware platforms with little or no change in the underlying codebase. The *Qt* framework first became publicly available in May 1995 which initially developed by *Haavard Nord* and *Eirik Chambe-Eng* (from Norwegian Institute of Technology in Trondheim and both graduated with master's degrees in computer science)[4]. Qt uses standard C++ with extensions including signals and slots that simplify handling of events, and this helps in development of both GUI and server applications which receive their own set of event information and should process them accordingly. Because of simplicity, robustness, native performance, cross-platform compatibility and both commercial and open source licenses, many organizations in many parts of the world use Qt.

1.7 Nucleic Acids:

Collectively referred polymers of nucleotides (*DNA* and *RNA*) as *Nucleic Acids* even in *Prokaryotic* (in some cases exist outside the nucleus) [10]. Each nucleotide consists of three components: a nitrogenous base (nucleobase), a pentose sugar (deoxyribose in case of *DNA* and ribose in case of *RNA*) and a phosphate group. There are two types of nitrogenous bases Purines and Pyrimidine's .the purines are *Adenine* and *Guanine* abbreviated A and G while Pyrimidine's are Cytosine, Thymine and Uracil abbreviated C, T, and U. Both DNA and RNA contain A, C, and G; only *DNA* contains the base T, whereas only *RNA* contains the base U.

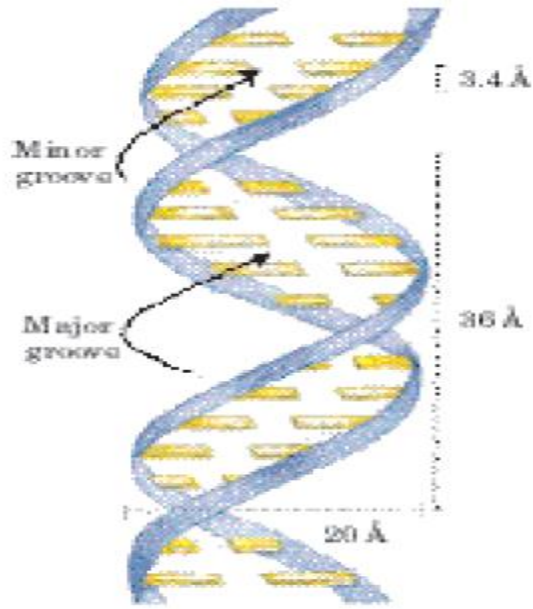


Figure 01.1 Watson - Crick Model for the structure of DNA [10].

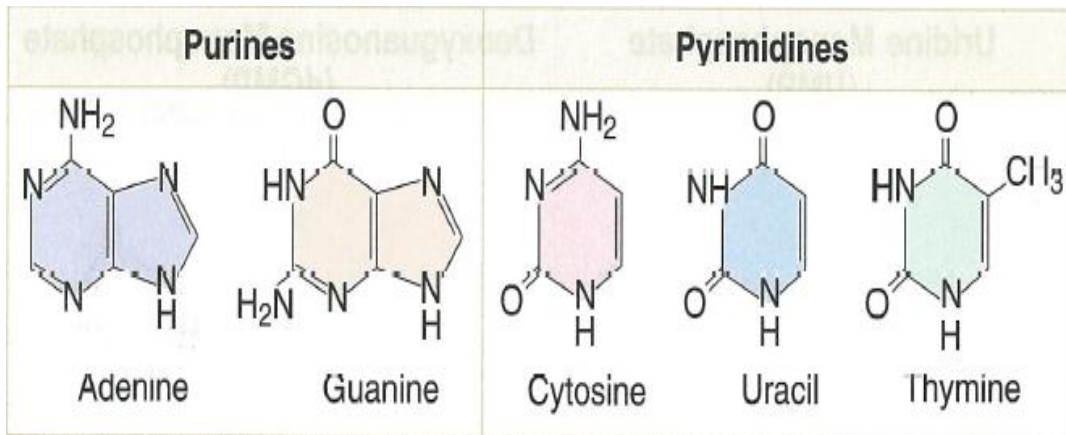


Figure 01.2 Bases commonly found in nucleic acids [11].

1.8 Proteins:

Proteins are built from twenty different amino acids (A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, M, Y). Each protein has its distinct amino-acid sequence and 3-dimensional structure. Proteins generated from DNA in process called gene expression. The process of gene expression divided into two steps, the first is transcription where the information coded in DNA is copied into a molecule of RNA whose bases are complementary to those of the DNA. The second is translation, where the information now encoded in RNA which translated into instructions for manufacturing a protein utilizing the ribosome protein machine.

1.9 Motivations:

There are many good reasons to prefer Qt/C++ to other languages for writing this library:

➤ **Portability:**

Qt is Platform-independent this means the code can be compiled under Windows or Linux running on x86 hardware or Solaris running on SPARC hardware.

➤ **Productivity:**

Qt comes with full set of effective built-in programming modules which help programming and save time such as QtGui, QtWebKit, QtNetwork, QtOpenGL, QSql and many others .

1.10 Objective:

The purpose of this thesis is to provide a programming library for scientists and Programmers working on problems in bioinformatics and computational biology. It may also appeal to programmers who want to improve their programming skills or programmers who have been working in bioinformatics and computational biology but are familiar with languages other than Qt/C++. A reasonable level of programming skill is presumed as is some familiarity with some of the basic tasks that need to be carried out in bioinformatics.

2 - Chapter Two:

String Processing Problems

DNA, RNA, and protein are represented as strings in bioinformatics for this reason string processing is the cornerstone in the field of bioinformatics and these problems take a variety of manifestations each of which has a specific meaning. This topic will shed some light on some traditional string problems such as: local sequence alignment problem, global sequence alignment problem, exact pattern matching problem, approximate pattern matching problem, finding all maximal palindromes problem, finding all tandem repeats problem, finding all tandem arrays problem, etc. There are quite rich researches for these problems. This thesis, will propose the major algorithms in this respect which implemented in BioQt.

2.1 Keywords:

Computational Biology; Bioinformatics ; Naive Algorithm ; Boyer-Moore ; Knuth–Morris–Pratt; Palindromes; Microsatellite; Manacher Algorithm; dynamic algorithms; Needleman-Wunsch; Smith-Waterman ; Longest Common Subsequence; Bit-Vector Algorithm.

2.2 Basic Definitions on String:

As mentioned earlier, DNA, RNA, and protein are represented as strings in bioinformatics. Therefore it's important to shed light on some basic definitions that are needed in string processing [12]:

2.2.1 Alphabet:

A set of allowable symbols. Examples of biosequence alphabets:

$\Sigma = \{A, C, G, T\}$ (DNA).

$\Sigma = \{A, C, G, U\}$ (RNA).

$\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$
(Proteins).

2.2.2 Sequence (or string):

A finite succession of characters chosen from an alphabet e.g.

ATCCGAACTTG from the DNA alphabet $\Sigma = \{A, C, G, T\}$.

2.2.3 Subsequence:

A sequence obtained from a sequence by removal of characters e.g.

TTT is a subsequence of ATATAT; AAAA is not a sub-sequence of ATATAT.

2.2.4 Substring:

A subsequence of consecutive characters e.g. TAC is a substring of

ACTACA, TAC is not a substring of ATGAC.

2.2.5 Length of string:

It's the number of characters in the string. It could be any non-negative integer. For example, if $\Sigma = \{A, C, G, T\}$, then ACGATGGGT is a string over Σ with length equals 9.

2.2.6 Prefix:

Substring containing the first character of a string, including the empty string e.g. in the string ACT, a prefix may be the empty string, A, AC, or ACT.

2.2.7 Proper prefix:

Any prefix of string except the string itself, a string of 1 letter has no proper prefix e.g. in the string ACT, a proper prefix are A, AC.

2.2.8 Suffix:

Substring containing the last character of a string e.g. in the string ACT, may be T, CT or ACT.

2.2.9 Proper Suffix:

Any suffix of string except the string itself e.g. in the string ACT, may be T or CT.

2.3 Microsatellite Repeats:

The term microsatellite was first coined by Litt and Luty [14]. Microsatellites are simple repeated motifs consisting of 1 to 6 base pairs contains a tandem of single, di, tri and tetra nucleotide repeat(e.g. a common repeat motif in birds is AC_n) [17] and they can be found in both intron and exon regions. Microsatellite repeats rarely occur within coding sequences but tri nucleotide repeats in or near genes are associated with certain inherited disorders (this will discussed as case study for Microsatellite Repeats in this thesis in next sections). The primary advantage of microsatellites as genetic markers is that they are inherited in a Mendelian fashion as co-dominant markers. DNA microsatellites are highly polymorphic and this means that the number of CA repeats for example varies between individuals and make them useable in linkage studies in gene mapping. This variation in repeat number is caused by incorrect base-pairing of the tandem repeats of the two complementary DNA strands during DNA replication (this phenomenon known as slipped strand mispairing) [15]. Furthermore, high abundance and a broad distribution throughout the genome have made microsatellites one of the most popular genetic markers for use in plant breeding programs(Gous Miah, Mohd Y.Rafii et.al) [16].

2.4 Objective:

Design Qt/C++ class to Detect Microsatellite Repeats in Sequences and investigated the position and frequencies of repeats.

2.5 Trinucleotide repeats disorder:

Also known as triplet repeats expansion disorders or codon reiteration disorders used as case study for testing BioQt microsatellite finder class. This a novel type of genome instability originally discovered in 1991 upon cloning the gene responsible for the fragile X syndrome, it has proved to be a general phenomenon responsible for a growing number of human neurological disorders [18]. This set of neurodegenerative disorders divided into two subgroups :

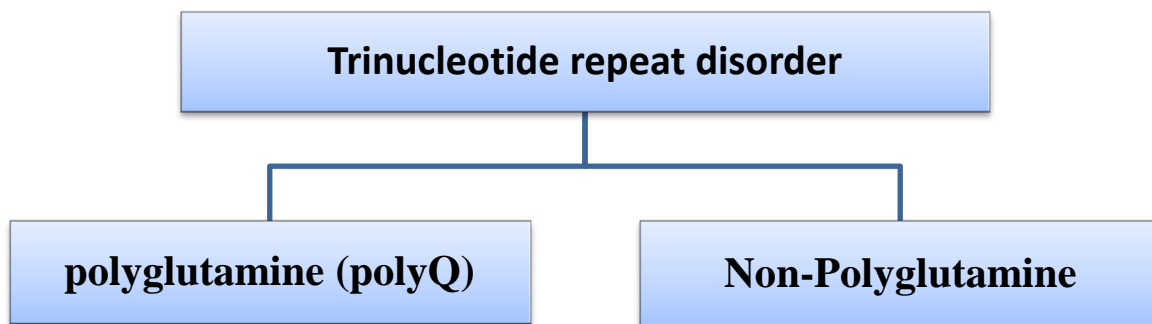


Figure: 2.1 Trinucleotide repeats disorder types.

2.5.1 Polyglutamine Diseases:

Caused by expanded cytosine-adenine-guanine (CAG) repeats situated in the coding regions of various human genes is linked to eight neurodegenerative diseases, including Huntington disease ,spinobulbar muscular atrophy,several spinocerebellar ataxias ,and dentatorubropallidolusian atrophy [18]. In all mentioned cases there is no effect on both transcription of target genes or translation of corresponding mRNAs [18] the only noticeable change is the repeat-

encoded polyglutamine stretches in the respective protein products lead to their self-aggregation and aggregation with other proteins(table 2.1).

Type	Gene	Normal PolyQ repeats	Pathogenic PolyQ repeats
DRPLA (Dentatorubropallidolusian atrophy)	ATN1 or DRPLA	6 - 35	49-88
HD (Huntington's disease)	HTT (Huntingtin)	6 - 35	36 - 250
SBMA (Spinal and bulbar muscular atrophy)	AR	9 - 36	38 - 62
SCA1 (Spinocerebellar ataxia Type 1)	ATXN1	6 - 35	49 - 88
SCA2 (Spinocerebellar ataxia Type 2)	ATXN2	14 - 32	33 - 77
SCA3 (Spinocerebellar ataxia Type 3 or Machado-Joseph disease)	ATXN3	12 - 40	55 - 86
SCA6 (Spinocerebellar ataxia Type 6)	CACNA1A	4 - 18	21 - 30
SCA7 (Spinocerebellar ataxia Type 7)	ATXN7	7 - 17	38 - 120
SCA17 (Spinocerebellar ataxia Type 17)	TBP	25 - 42	47 - 63

Table 2.1 Polyglutamine (PolyQ) Diseases

2.5.2 Non-Polyglutamine Diseases:

Beyond PolyQ group there is another subset of disorders termed as non-polyglutamine diseases which also fall under the category of trinucleotide repeat disorders. The researchers have identified six non-

polyglutamine each which exhibit a unique repeated codon , These diseases has relatively minor resemblance to each other's. Noticeably none of them appear to have any strong similarity to Huntington's disease or the other PolyQ diseases (table 2.2).

Type	Gene	Codon	Normal/wild type	Pathogenic
FRAXA (Fragile X syndrome)	<i>FMRI</i> , on the X-chromosome	CGG	6 - 53	230+
FXTAS (Fragile X-associated tremor/ataxia syndrome)	<i>FMRI</i> , on the X-chromosome	CGG	6 - 53	55-200
FRAXE (Fragile XE mental retardation)	AFF2 or FMR2, on the X-chromosome	CCG	6 - 35	200+
FRDA (Friedreich's ataxia)	FXN or X25, (frataxin— reduced expression)	GAA	7 - 34	100+
DM (Myotonic dystrophy)	<u>DMPK</u>	CTG	5 - 37	50+
SCA8 (Spinocerebellar ataxia Type 8)	OSCA or SCA8	CTG	16 - 37	110 - 250
SCA12 (Spinocerebellar ataxia Type 12)	PPP2R2B or SCA12	nnn On 5' end	7 - 28	66 - 78

Table 2.2 Non-Polyglutamine Diseases.

2.6 Material:

- C++ compiler gnu GCC for Unix/mac or VC++ for MS windows.
- Qt SDK.
- Any Computer platforms (Pc/Mac or UNIX).

2.7 Methods:

FindMicrosatelliteRepeats class is written in Qt/C++ and calculates Microsatellite Repeats positions and frequencies uses sliding window algorithm.

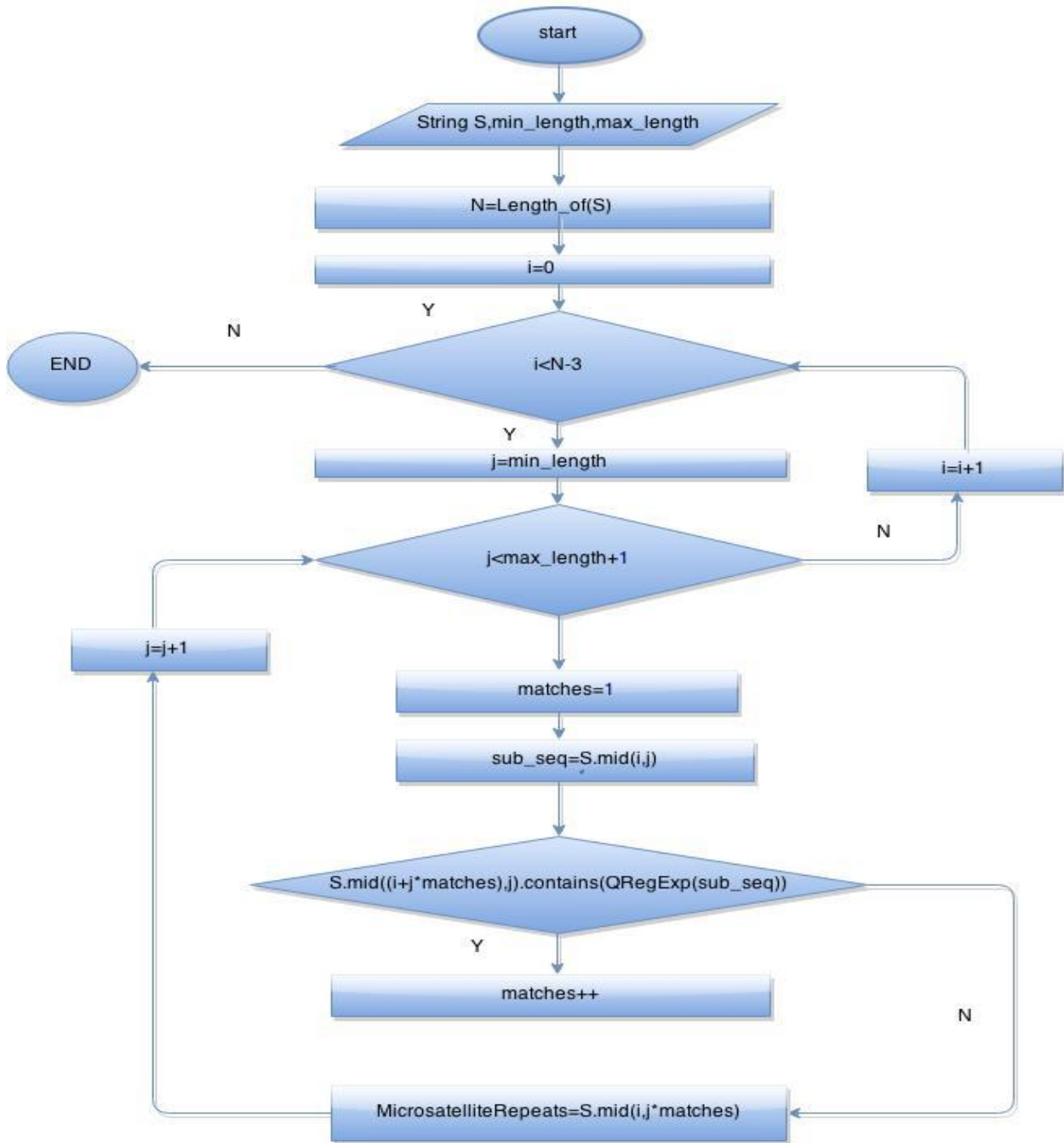


Figure 2.2 Microsatellites Repeats Algorithm Flowchart.

2.8 Palindromes:

A palindrome, in the literary sense, refers to a word or a phrase that reads the same in both directions, i.e. when it is read in forward and reverse. One of the oldest palindromes known is the phrase “Sator arepo tenet opera rotas” [19]. But in the biological context definition of palindromes is slightly differs, in case of proteins there is no difference compared to regular English language definition of palindrome, in contrast the case of either DNA or RNA shows evident difference. For a nucleotide sequence to be considered as a palindrome, its complementary strand must read the same in the opposite direction i.e. the sequence 5`-CGATCG-3` is considered a palindrome since its reverse complement 3`-GCTAGC-5` reads the same (David Roy Smith et. al) [20]. Palindromes can be even or odd. An odd palindrome is one in which there is no asymmetry in the center or mismatch in any part of the sequence, such that its reverse compliment reads the same as the original sequence itself.

2.9 Importance of Palindromic sequence:

Palindromic sequences are present in the genomes of all organisms, these motifs considerably participate in a various cellular processes regulation [24], but on the other hand they are also responsible for a numerous of genetic instability. At the nucleic acids levels palindromic sequences tend to create hairpin loops (also known as stem-loops) [21]. No stochastic distribution of DNA palindromic patterns have been observed in cancer cells and are attributed to mycogene amplification [22, 23]. Considering the incidence of palindromes at the genomic level this thesis implements a class that can identify, locate and count palindromes in a given sequence in a strictly defined way. According to palindrome counts were

significantly different from those in the randomly generated DNA sequences (dummy sequence) use of real sequence well impact on a case study (discussed in details in further sections of this thesis).

2.10 Objective:

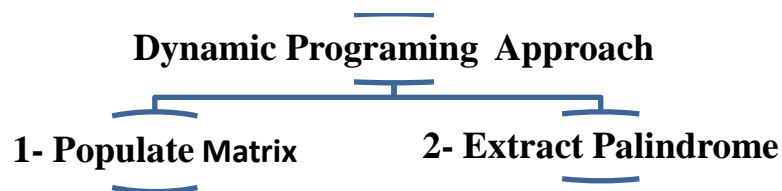
Implement Qt/C++ class to detect Palindrome Subsequence.

2.11 Material:

- C++ compiler gnu GCC for Unix/mac or VC++ for MS windows
- Qt SDK
- Any Computer platforms (Pc/Mac or UNIX).

2.12 Methods:

Dynamic programming approach.



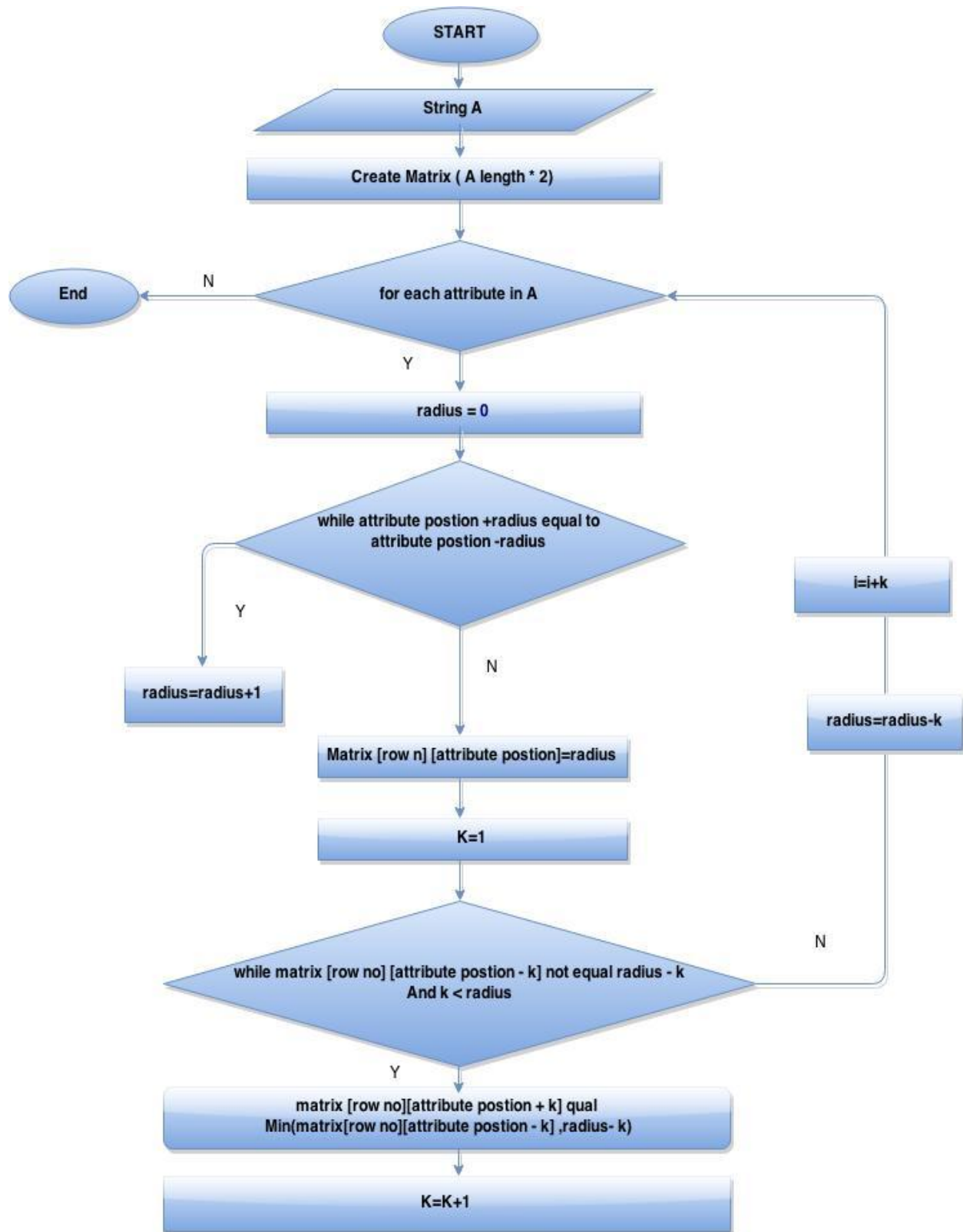


Figure 2.3 Palindromes Finder Populate Matrix Flowchart.

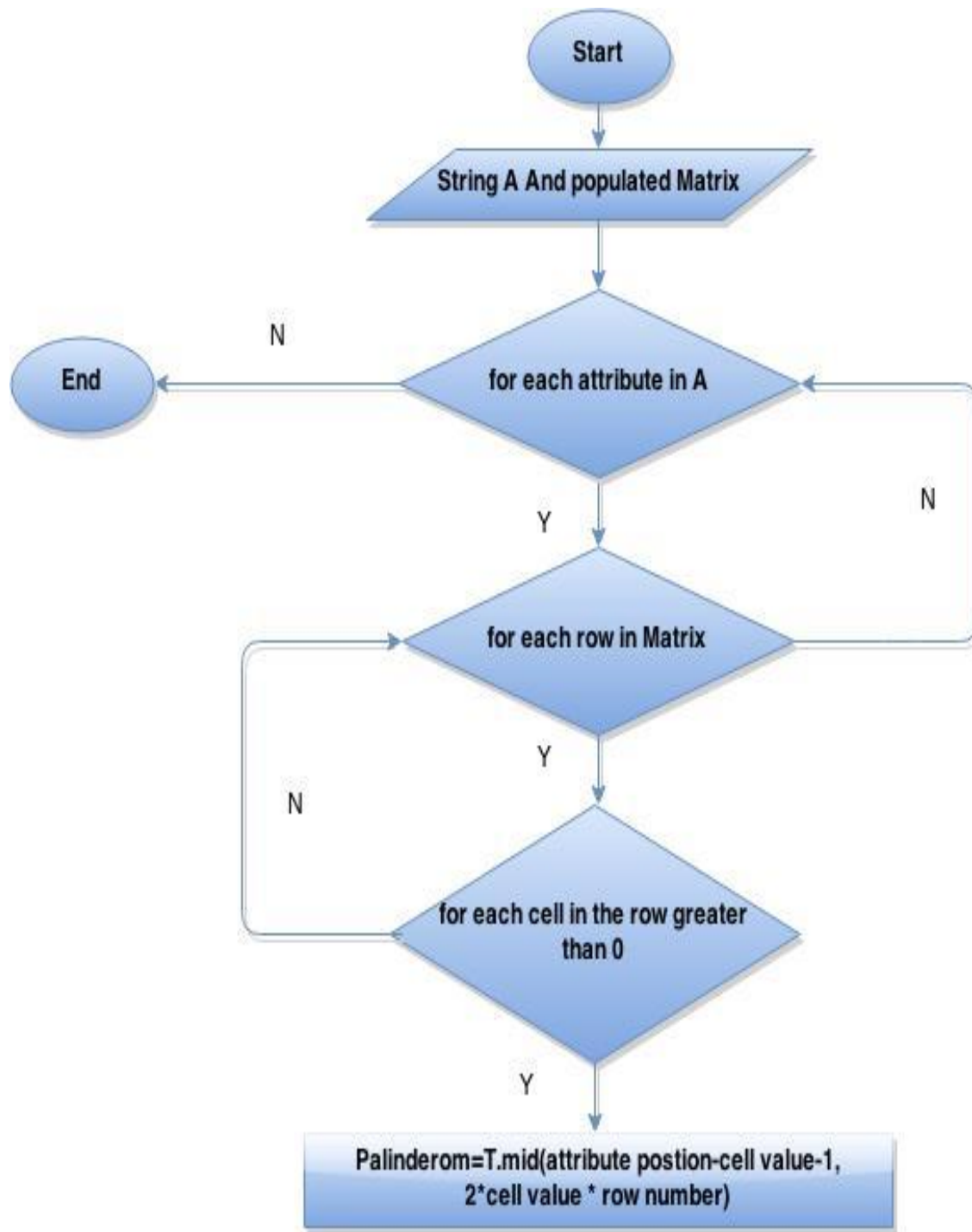


Figure 2.4 Palindromes Finder Extract Palindromes Flowchart.

2.13 Exact String Matching:

One of the most popular and well-studied problem is the exact string matching. The task is to find out the occurrences of a particular string pattern in a long text and according to unprecedented rapidity of biological sequence databases growth rate the demands for efficient and high performance computational algorithms for comparing and searching biological sequences have also increased. This thesis will introduce several traditional algorithms for exact string matching and port them to Qt/C++ and study their experimental performance and compare their efficiency. Algorithms of this type are greedy in the sense that the pattern is moved forward after the first character mismatch of an alignment is observed. Shifts by occurrence shift may then be unnecessarily short, if shifting is based on a single character. On the other hand the probability of having the algorithm to enter to the slow loop is almost always higher when the decision is based on a single character.

2.14 Objective:

Implement Qt/C++ class to solve the exact string matching.

2.15 Material:

- C++ compiler gnu GCC for Unix/mac or VC++ for MS windows.
- Qt SDK.
- Any Computer platforms (Pc/Mac or UNIX).

2.16 Methods:

1. Brute Force (Naive) algorithm.
2. Knuth-Morris-Pratt (KMP) algorithm.
3. Boyer-Moore Algorithm.
4. Baeza-Yates–Gonnet (shift-or/and) algorithm.

2.17 Brute Force:

Definitely the first brainstorm solution for solving the exact string matching is a brute force algorithm (exhaustive search) it's a very general and trivial problem solving technique this method try all possible solutions by enumerating all possible candidates and checking whether each candidate satisfies the problem's statement. Then scanning is done from left to right. As shifting is done at each step it gives less efficiency.

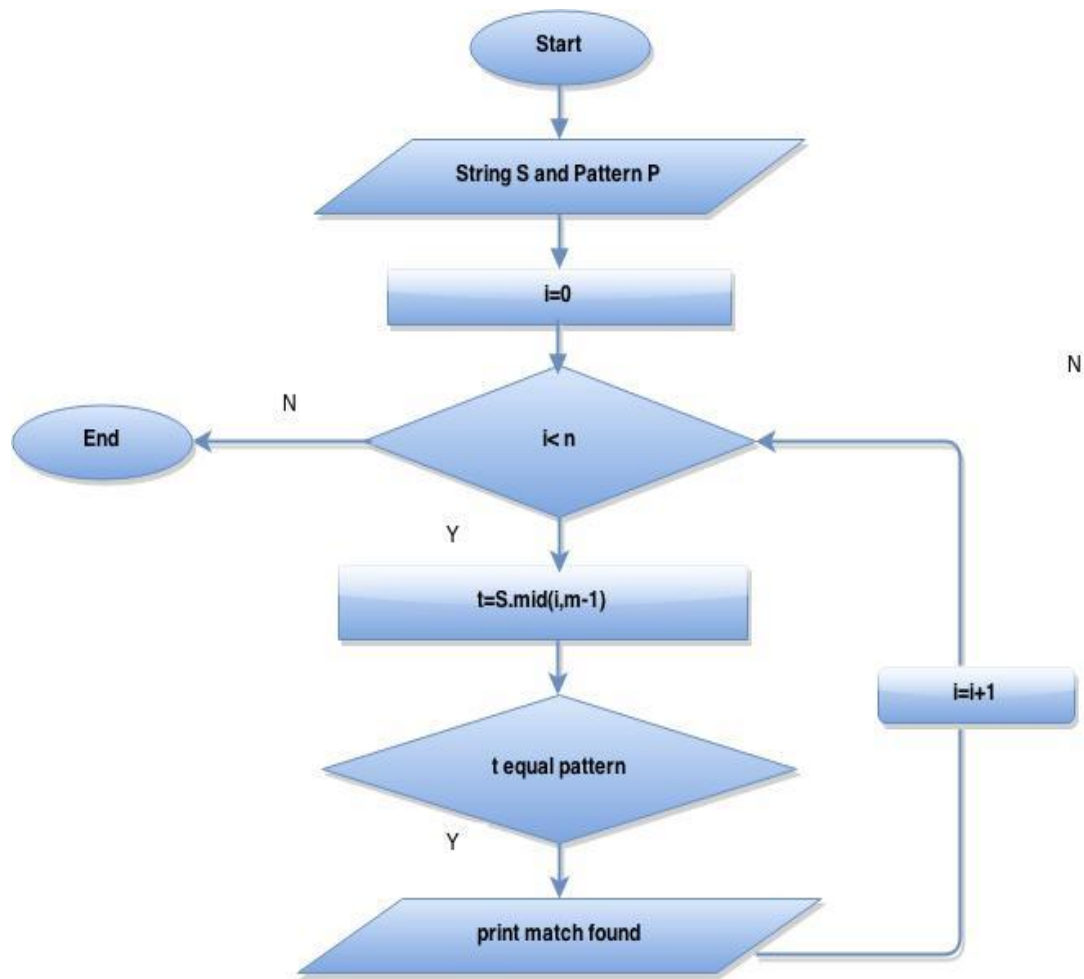
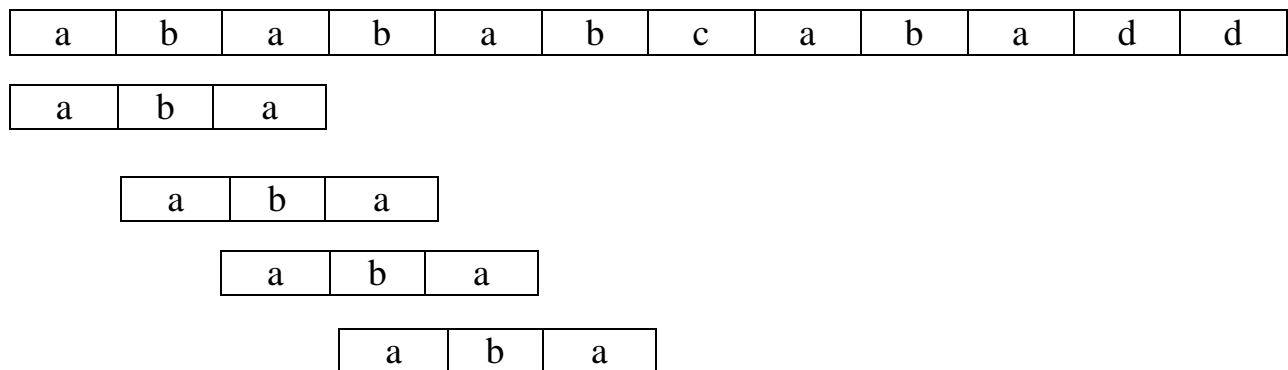


Figure 2.5 Brute Force Algorithm Flowcharts.

2.17.1 Brute Force Visualization:



It's obvious the algorithm is simple and *redundant* e.g. if $m=3$ the algorithm will compare positions: 0 1 2 , 1 2 3 etc. to avoid redundancy a smarter solution is needed and this can be done by using either Boyer Moor's or KMP. Algorithms of this type are greedy in the sense that the pattern is moved forward after the first character mismatch of an alignment is observed. Shifts by occurrence shift may then be unnecessarily short if shifting is based on a single character. On the other hand, the probability of having the algorithm to enter to the slow loop is almost always higher when the decision is based on a single character.

2.18 The Knuth-Morris-Pratt (KMP) Algorithm:

This algorithm was conceived by Donald Knuth and Vaughan Pratt and independently by James H. Morris in 1977 [25]. Knuth, Morris and Pratt discovered first linear time string-matching algorithm by analysis of the Naive algorithm. It keeps the information that Naive approach wasted gathered during the scan of the text. By avoiding this waste of information it achieves a running time of $O(m + n)$. The implementation of Knuth-Morris-Pratt algorithm is efficient because it minimizes the total number of comparisons of the pattern against the input string.

2.18.1 The prefix-function π :

- It preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself.
- It is defined as the size of the largest prefix of $P[0..j - 1]$ that is also a suffix of $P[1..j]$.

- It also indicates how much of the last comparison can be reused if it fails.
- It enables avoiding backtracking on the string "S".

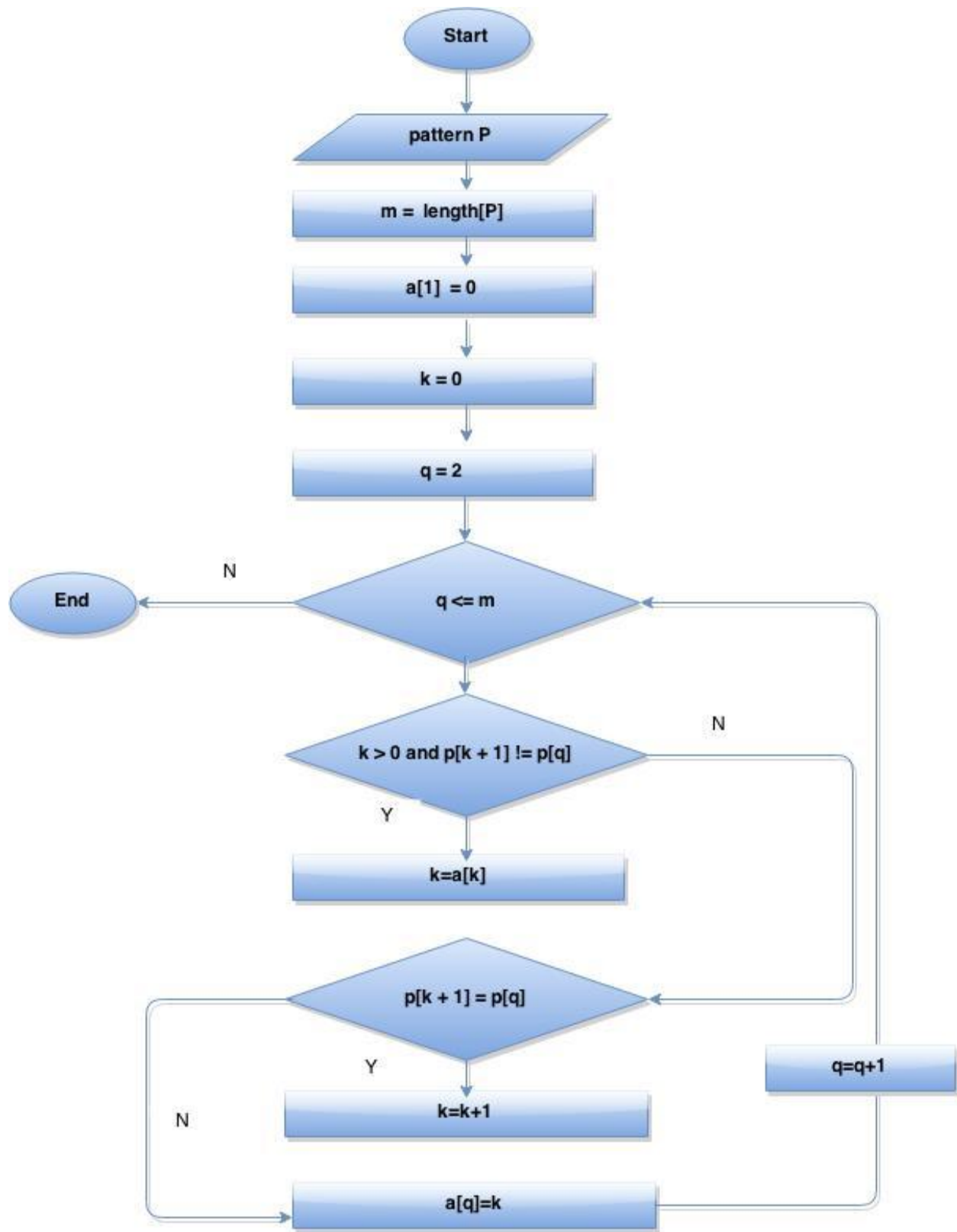


Figure 2.6 KMP prefix function Flowchart.

2.18.2 To compute π for the pattern 'p':

Pattern	a	b	a	b	a	c	a
----------------	----------	----------	----------	----------	----------	----------	----------

Initially: $m = \text{length}[p] = 7$.

$$\pi[1] = 0.$$

$$k = 0.$$

Where m , $\pi[1]$, and k are the length of the pattern, prefix function and initial potential value respectively.

Step 1: $q = 2$, $k = 0$ and $\pi[2] = 0$.

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0					

Step 2: $q = 3$, $k = 0$ and $\pi[3] = 1$.

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1				

Step 3: $q = 4$, $k = 1$ and $\pi[4] = 2$.

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2			

Step 4: $q = 5$, $k = 2$ and $\pi [5] = 3$.

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3		

Step 5: $q = 6$, $k = 3$ and $\pi [6] = 1$.

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3	1	

Step 6: $q = 7$, $k = 1$ and $\pi [7] = 1$.

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3	1	1

2.18.3 Find matches algorithm:

Step 1: Initialize the input variables:

- $n = \text{Length of the Text.}$
- $m = \text{Length of the Pattern.}$
- $\pi = \text{Prefix-function of pattern (p).}$
- $q = \text{Number of characters matched.}$

Step 2: Define the variable:

- $q=0$, the beginning of the match.

Step 3: Compare the first character of the pat tern with first character of Text. If match i s not found, substitute the value of $\pi[q]$ to q.

I f match is found, and then increment the value of q by 1.

Step 4: Check whether all the pat tern elements are matched with the text elements if not, repeat the search process. If yes, print the number of shifts taken by the pat -tern.

Step 5: look for the next match.

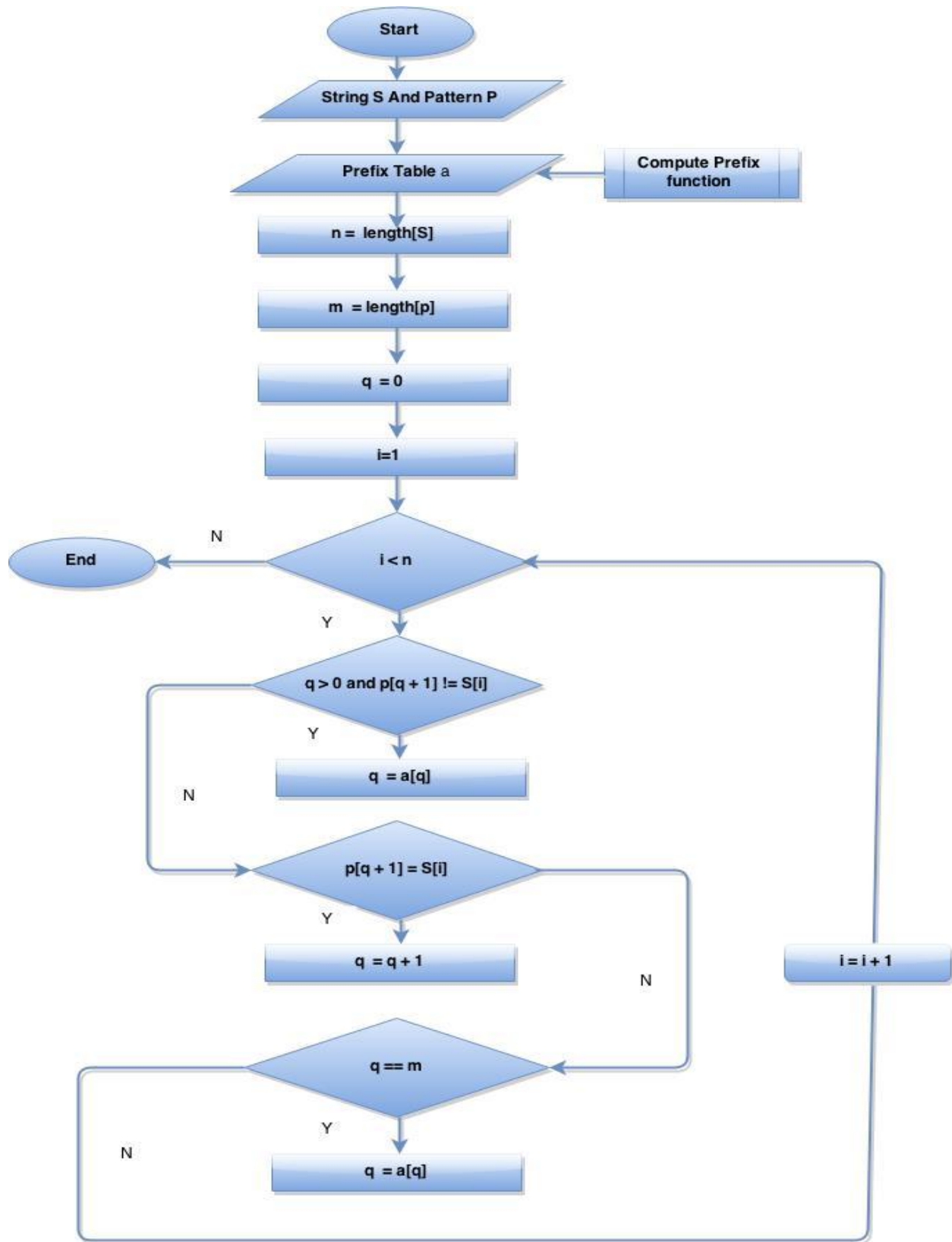


Figure 2.7 KMP find matches algorithm.

String	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b
--------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Pattern	a	b	a	b	a	c	a
---------	---	---	---	---	---	---	---

Execute the KMP algorithm to find whether 'p' occurs in 'S'.

Initially: $n = \text{size of } S = 15$, $m = \text{size of } p = 7$.

Step 1: $i = 1$, $q = 0$.

Comparing $p[1]$ with $S[1]$.

String	<u>b</u>	a	c	b	a	b	a	b	a	b	a	c	a	a	b
--------	----------	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Pattern	<u>a</u>	b	a	b	a	c	a
---------	----------	---	---	---	---	---	---

$P[1]$ does not match with $S[1]$. 'P' will be shifted one position to the right.

Step 2: $i = 2$, $q = 0$.

Comparing $p[1]$ with $S[2]$.

String	b	<u>a</u>	c	b	a	b	a	b	a	b	a	c	a	a	b
--------	---	----------	---	---	---	---	---	---	---	---	---	---	---	---	---

Pattern	<u>a</u>	b	a	b	a	c	a
---------	----------	---	---	---	---	---	---

Step 3: $i = 3, q = 1$.

Comparing $p[2]$ with $S[3]$ $p[2]$ does not match with $S[3]$.

String	b	a	<u>c</u>	b	a	b	a	b	a	b	a	c	a	a	b
--------	---	---	----------	---	---	---	---	---	---	---	---	---	---	---	---

Pattern	a	<u>b</u>	a	b	a	c	a
---------	---	----------	---	---	---	---	---

Backtracking on p , comparing $p[1]$ and $S[3]$.

Step 4: $i = 4, q = 0$.

Comparing $p[1]$ with $S[4]$ $p[1]$ does not match with $S[4]$.

String	b	a	c	<u>b</u>	a	b	a	b	a	b	a	c	a	a	b
--------	---	---	---	----------	---	---	---	---	---	---	---	---	---	---	---

Pattern	<u>a</u>	b	a	b	a	c	a
---------	----------	---	---	---	---	---	---

Step 5: $i = 5, q = 0$.

Comparing $p[1]$ with $S[5]$.

String	b	a	c	b	<u>a</u>	b	a	b	a	b	a	c	a	a	b
--------	---	---	---	---	----------	---	---	---	---	---	---	---	---	---	---

Pattern	<u>a</u>	b	a	b	a	c	a
---------	----------	---	---	---	---	---	---

Step 6: $i = 6, q = 1$.

Comparing $p[2]$ with $S[6]$ $p[2]$ matches with $S[6]$.

String	b	a	c	b	<u>a</u>	b	a	b	a	b	a	c	a	a	b
--------	---	---	---	---	----------	---	---	---	---	---	---	---	---	---	---

Pattern	<u>a</u>	b	a	b	a	c	a
---------	----------	---	---	---	---	---	---

Step 7: $i = 7, q = 2$.

Comparing $p[3]$ with $S[7]$ $p[3]$ matches with $S[7]$.

String	b	a	c	b	<u>a</u>	<u>b</u>	a	b	a	b	a	c	a	a	b
--------	---	---	---	---	----------	----------	---	---	---	---	---	---	---	---	---

Pattern	<u>a</u>	<u>b</u>	a	b	a	c	a
---------	----------	----------	---	---	---	---	---

Step 8: $i = 8, q = 3$.

Comparing $p[4]$ with $S[8]$ $p[4]$ matches with $S[8]$.

String	b	a	c	b	<u>a</u>	<u>b</u>	<u>a</u>	b	a	b	a	c	a	a	b
--------	---	---	---	---	----------	----------	----------	---	---	---	---	---	---	---	---

Pattern	<u>a</u>	<u>b</u>	<u>a</u>	b	a	c	a
---------	----------	----------	----------	---	---	---	---

Step 9: $i = 9, q = 4$.

Comparing $p[5]$ with $S[9]$ $p[5]$ matches with $S[9]$.

String	b	a	c	b	<u>a</u>	<u>b</u>	<u>a</u>	<u>b</u>	a	b	a	c	a	a	b
--------	---	---	---	---	----------	----------	----------	----------	---	---	---	---	---	---	---

Pattern	<u>a</u>	<u>b</u>	<u>a</u>	<u>b</u>	a	c	a
---------	----------	----------	----------	----------	---	---	---

Step 10: $i = 10, q = 5$.

Comparing $p[6]$ with $S[10]$ $p[6]$ doesn't matches with $S[10]$.

String	b	a	c	b	<u>a</u>	<u>b</u>	<u>a</u>	<u>b</u>	<u>a</u>	b	a	c	a	a	b
--------	---	---	---	---	----------	----------	----------	----------	----------	---	---	---	---	---	---

Pattern	<u>a</u>	<u>b</u>	<u>a</u>	<u>b</u>	<u>a</u>	c	a
---------	----------	----------	----------	----------	----------	---	---

Back tracking on p , comparing $p[4]$ with $S[10]$ because after mismatch

$$q = \pi[5] = 3.$$

Step 11: $i = 11, q = 4$.

Comparing $p[5]$ with $S[11]$.

String	b	a	c	b	a	b	<u>a</u>	<u>b</u>	<u>a</u>	<u>b</u>	<u>a</u>	c	a	a	b
--------	---	---	---	---	---	---	----------	----------	----------	----------	----------	---	---	---	---

Pattern	<u>a</u>	<u>b</u>	<u>a</u>	<u>b</u>	<u>a</u>	c	a
----------------	----------	----------	----------	----------	----------	---	---

Step 12: $i = 12, q = 5$.

Comparing $p[6]$ with $S[12]$ $p[6]$ matches with $S[12]$.

String	b	a	c	b	a	b	<u>a</u>	<u>b</u>	<u>a</u>	<u>b</u>	<u>a</u>	<u>c</u>	<u>a</u>	a	b
---------------	---	---	---	---	---	---	----------	----------	----------	----------	----------	----------	----------	---	---

Pattern	<u>a</u>	<u>b</u>	<u>a</u>	<u>b</u>	<u>a</u>	c	a
----------------	----------	----------	----------	----------	----------	---	---

Step 13: $i = 13, q = 6$.

Comparing $p[7]$ with $S[13]$ $p[7]$ matches with $S[13]$.

String	b	a	c	b	a	b	<u>a</u>	<u>b</u>	<u>a</u>	<u>b</u>	<u>a</u>	<u>c</u>	<u>a</u>	a	b
---------------	---	---	---	---	---	---	----------	----------	----------	----------	----------	----------	----------	---	---

Pattern	<u>a</u>	<u>b</u>	<u>a</u>	<u>b</u>	<u>a</u>	<u>c</u>	<u>a</u>
----------------	----------	----------	----------	----------	----------	----------	----------

Pattern 'p' has been found to completely occur in string 'S'.

The total number of shifts that took place for the match to be found is:

$$i - m = 13 - 7 = 6 \text{ shifts.}$$

$O(m)$ - It is to compute the prefix function values.

$O(n)$ - It is to compare the pattern to the text.

Total of $O(n + m)$ run time.

2.18.4 Advantage:

- Best known for linear time for exact matching.
- Compares from left to right.
- Shifts more than one position.
- Preprocessing approach of Pattern to avoid trivial
- Comparisons.
- Avoids recomputing matches.

2.18.5 Disadvantages:

- Doesn't work as well as the size of the alphabets increases. By which more chances of mismatch occurs.

2.19 Boyer-Moore Algorithm:

It coined by Bob Boyer and J.Strother Moore in 1977 [26], at that time it considered as the most efficient string matching algorithm. This algorithm performs comparison task in reverse order from right to the left of the pattern and did not require the whole pattern to be searched in case of a mismatch (instead of all previous algorithms which fall under the category of exact string matching). Since 1977 this algorithm subjected to a quite number of improvements

to increase both efficiency and accuracy and as result of that there are too many variations of this algorithm e.g. Boyer-Moore Smith (MBS), Turbo Boyer Moore (TBM), Two Way algorithm (TW), Berry Ravindran algorithm (BR), Reverse Colussi algorithm (RC) and Sunday algorithm .

2.19.1 Analysis of Boyer-Moore Algorithm:

Boyer-Moor algorithm is one of heuristic (machine learning) algorithms and performs its search task by matching the pattern with text from right to the left and shifting the pattern from left to right the shifting procedure governed by rules :

- If mismatch reported use knowledge of the mismatched text character to skip alignments and this called Bad character Heuristic (Easy Case).
- If some charters matched use knowledge of the matched characters to skip alignments and this called Good suffix Heuristic.
- Try alignments in one direction, and then try character comparisons in opposite direction (long skip).

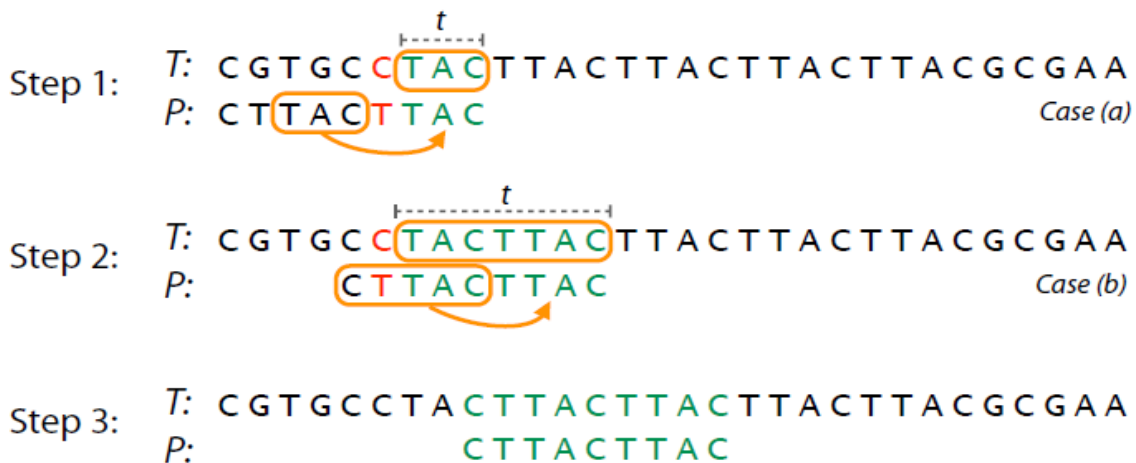
2.19.2 Bad character Heuristic:

Upon mismatch, let b be the mismatched character in T . Skip alignments until (a) b matches its opposite in P , or (b) P moves past b .



2.19.3 Good suffix Heuristic:

Let t be the substring of T that matched a suffix of P . Skip alignments until (a) t matches opposite characters in P , or (b) a prefix of P matches a suffix of t , or (c) P moves past t , whichever happens first



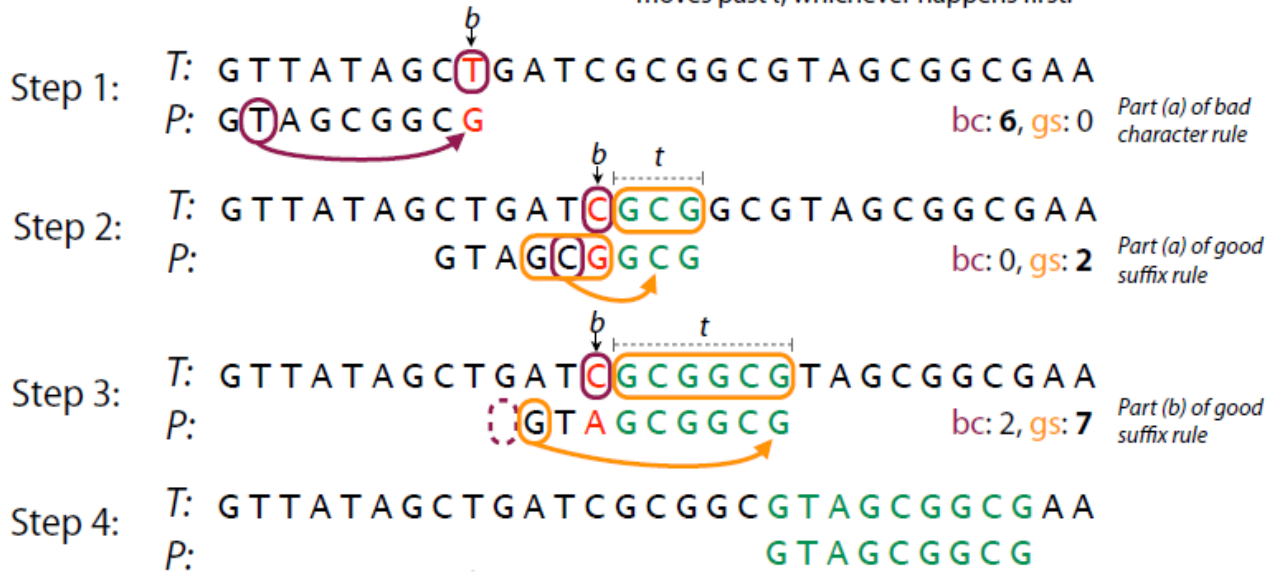
After each alignment, use bad character or good suffix rule, whichever skips more

Bad character rule:

Upon mismatch, let b be the mismatched character in T . Skip alignments until (a) b matches its opposite in P , or (b) P moves past b .

Good suffix rule:

Let t be the substring of T that matched a suffix of P . Skip alignments until (a) t matches opposite characters in P , or (b) a prefix of P matches a suffix of t , or (c) P moves past t , whichever happens first.



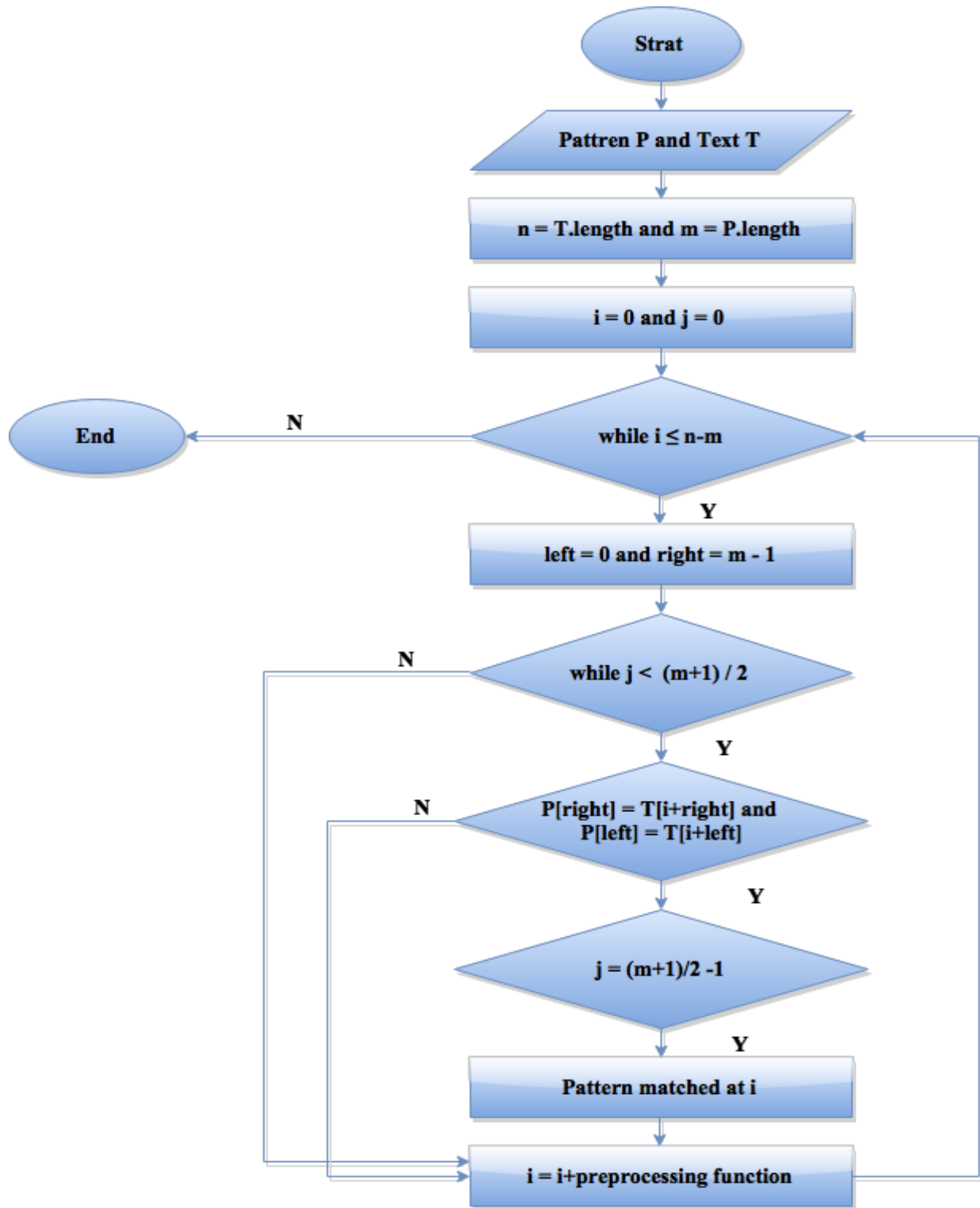


Figure 2.8 Boyer-Moor find matches algorithm.

2.19.4 Advantages:

- The both good-suffix and bad-char combined provides a good shift value as maximum of two is taken as shift value.

2.19.5 Disadvantages:

- The preprocessing of good-suffix is complex to implement and understand.
- Bad-char of mismatch character may give small shift, if mismatch after many matches.

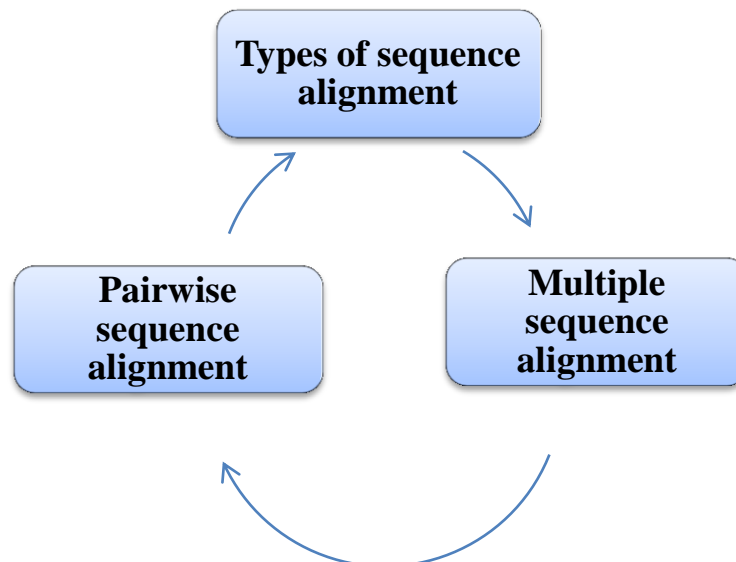
3 - Chapter Three:

String Matching Evaluation Methods

Identifying patterns among biological sequences was the title for a lot of researches in bioinformatics. Evaluation of string matching or approximate string matching (often colloquially referred to as fuzzy string searching) is performed by implementations of different algorithms on the theoretical side, some of the famous algorithms used in sequences comparison such as: Longest Common Substring (LCS), Subsequence (LCSS), Global alignment (Needleman-Wunsch) and Local alignment (Smith-Waterman) algorithms. Many biological machines share very similar gene sequences while some regions of sequences vary from spice to spice or even among individuals from the same domain. The comparison committed by recognizing the regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. i.e. the similarity shows that the sequences share a common ancestral sequence. As a result similar sequences have similar functionality. Such sequences are said to be homologous. Furthermore the knowledge of a DNA sequence and gene analysis can be used in several biological, medical and agricultural research fields such as: possible disease or abnormality diagnoses, forensics, pattern matching, biotechnology, etc. The analysis and comparison studies for DNA sequences can be used to detect possible errors or abnormality in a DNA sequence and predict the function of a specific gene and compare it with other similar genes from same or different organisms. Each newly discovered DNA sequence is classified according to its Similarity with known DNA sequences.

3.1 Sequence Alignment:

Sequence alignment process means put two or more sequences above each other and study the similarity between them this study done in terms of quantity. These comparisons studies can be used to extract information such as: detect homology, evolutionary divergence, predict of function, origin of diseases and model 3D-structure. There are two methods to do alignment either pairwise or multiple sequence alignment.



3.2 Biological Motivation

- Comparing or retrieving DNA/Protein sequences in databases.
- Comparing two or more sequences for similarities.
- Finding patterns within a protein or DNA sequence.
- Tracking the evolution of sequences.

3.3 Terminology:

3.3.1 Identity:

Proportion of pairs of identical residues between two aligned sequences. Generally expressed as a percentage. This value strongly depends on how the two sequences are aligned.

3.3.2 Similarity:

Proportion of pairs of similar residues between two aligned sequences. If two residues is similar is determined by a substitution matrix.

This value also depends strongly on how the two sequences are aligned, as well as on the substitution matrix used.

3.3.3 Homology:

Two sequences are homologous if and only if they have a common ancestor. There is no such thing as a level of homology (It's either yes or no).

3.4 Pairwise Alignment:

Pairwise sequence alignment methods are used to find the best matching piecewise either local or global alignments of two query sequences. This method implemented only between two sequences at a time. This method is efficient and often used between sequences that do not need extreme precision e.g. query sequences with high similarity. There are four major methods to do pairwise sequence alignment:

- Manually (by using white board or any word processing application).
- dot plot method this method introduced by Gibbs and McIntyre 1970 [28]

The idea of this method is:

- ❖ Sequence (A) is listed across the top of the matrix and the other (B) is listed down the left side.
- ❖ Starting from the first character in B, one moves across the page keeping in the first row and placing a dot in many columns where the character in A is the same.
- ❖ The process is continued until all possible comparisons between A and B is made.
- ❖ Any region of similarity is revealed by a diagonal row of dots.
- ❖ Isolated dots not on diagonal represent random matches.

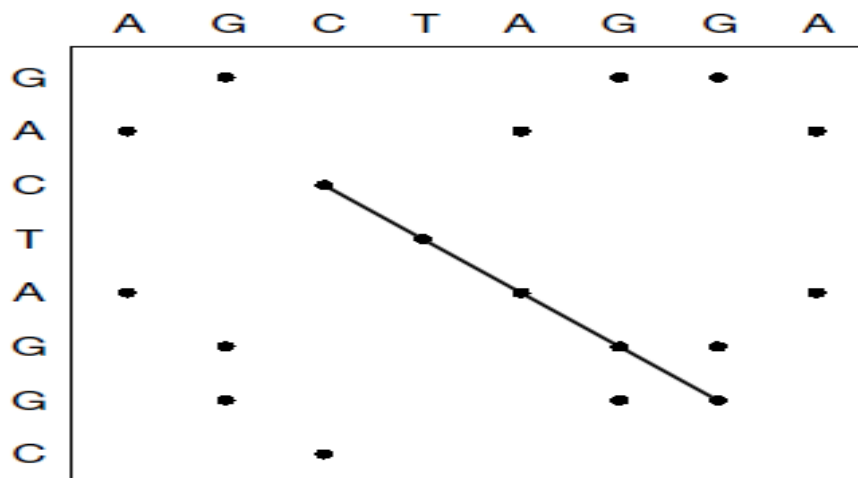


Figure 3.1 a simple dot matrix comparison of two DNA sequences AGCTAGGA and GACTAGGC [29].

- Dynamic Programming methods (Rigorous mathematical) this approach is slow but optimal and uses two algorithms Needleman-Wunsch algorithm (for global pairwise alignment) and Smith-Waterman algorithm (for local

pairwise alignment). These two algorithms will be discussed in details in this thesis.

- Heuristic algorithms (k-tuple method) this approach is faster but approximate and uses either BLAST or FASTA.

3.5 Multiple Sequence Alignment:

It's an extension of pairwise alignment in this approach more than two sequences are involved at a time. Multiple alignments are often used in phylogenetic analysis, detection of homology between a newly sequenced gene and an existing gene family, prediction of protein structure and demonstration of homology in multigene families. The most widely used progressive alignment algorithm is currently CLUSTALW.

3.6 Pairwise Sequence Alignment by Dynamic Programming:

As mentioned before in section 3.4 there is more than one method for performing Pairwise Sequence Alignment. This thesis will focus on DP (Dynamic Programming) methods which first introduced by Richard Bellman in 1953 to study multistage decision problems [30]. The idea behind this style of problem solving is breaking down the big problem to smaller sub problems (often many of these sub problems are really the same) by solving each sub problems only once this will reducing the number of computations. This approach is especially useful when the number of repeating sub problems grows exponentially as a function of the size of the input. In this method each pair of characters is compared in the two

sequences and generates an alignment. This alignment will include matched and mismatched characters and gaps in the two sequences that are positioned so that the number of matches between identical or related characters is the maximum possible. The dynamic programming algorithm provides a reliable computational method for aligning DNA and protein sequences. The method has been proven mathematically to produce the best or optimal alignment between two sequences under a given set of match conditions. Optimal alignments provide useful information to biologists concerning sequence relationships by giving the best possible information as to which characters in a sequence should be in the same column in an alignment and which are insertions in one of the sequences (or deletions on the other). This information is important for making functional, structural and evolutionary predictions on the basis of sequence alignments. There are two important algorithms used in DP pairwise sequence alignment:

- Needleman and Wunsch Algorithm (for pairwise global sequence alignment).
- Smith and Waterman algorithm (for pairwise local sequence alignment).

3.7 Global Alignment (Needleman-Wunsch Algorithm):

The dynamic programming method for global sequence alignment was described by Needleman and Wunsch (1970) [31] and proved mathematically and extended to include an improved scoring system by Smith and Waterman [32]. This problem focusing on how to find the best alignment based on measurements of similarity which uses certain scoring functions (table 3.1). The optimal score at each matrix position is calculated by adding the current match score to previously scored positions and subtracting gap penalties if applicable. Each matrix position may

have a positive or negative score or 0. The Needleman-Wunsch algorithm will maximize the number of matches between the sequences along the entire length of the sequences. Gaps may also be present at the ends of sequences in case there is extra sequence left over after the alignment. These end gaps are often but not always given a gap penalty.

Case	Score
Match	+1
Mismatch	-1
Gap	-2

Table 3.1 scoring function.

To build up an optimal alignment using previous solutions for optimal alignments of smaller subsequences for given two sequences $x = (x_1, x_2, \dots, x_i)$ and $y = (y_1, y_2, \dots, y_j)$.

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + 1, & x_i = y_i \text{ (match)} \\ F(i-1, j-1) - 1, & x_i \neq y_i \text{ (substitution)} \\ F(i-1, j) - 2, & \text{align } x_i \text{ with a gap} \\ F(i, j-1) - 2, & \text{align } y_j \text{ with a gap} \end{cases}$$

in which $F(i, j)$ equals the best score of the alignment of the two prefixes (x_1, x_2, \dots, x_i) and (y_1, y_2, \dots, y_j) . This will be done recursively by setting $F(0, 0) = 0$ and then computing $F(i, j)$ from $F(i-1, j-1)$, $F(i-1, j)$ and $F(i, j-1)$.

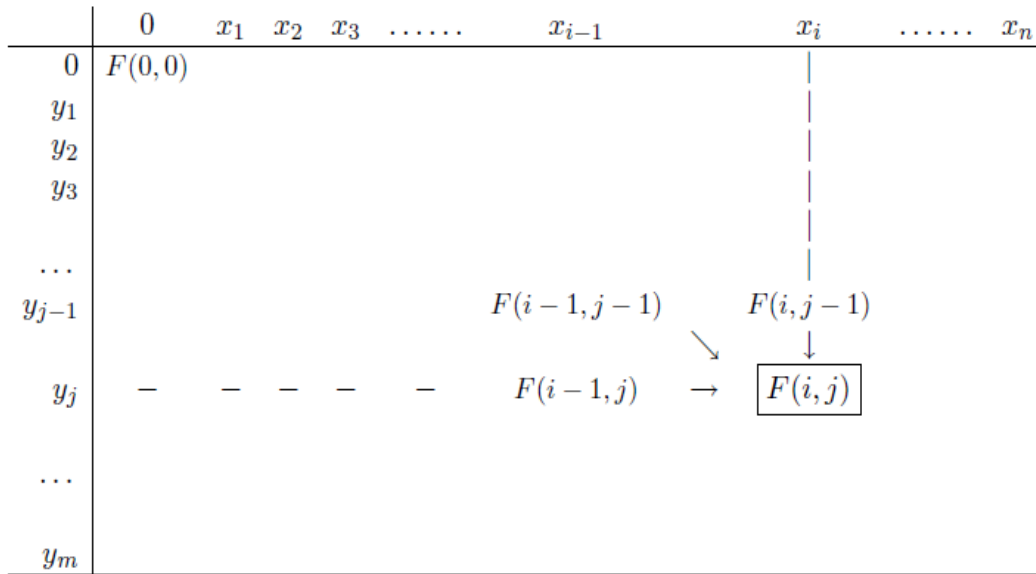


Figure 3.2 Calculate Best Score.

For example to align two sequences ACCTGA and AGCTA the scoring matrix will be as table 3.2. After the matrix fill step the maximum alignment score for the two test sequences is 1. The traceback step determines the actual alignment(s) that result in the maximum score (table 3.3).

	-	A	C	C	T	G	A
-	0	-2	-4	-6	-8	-10	-12
A	-2	1	-1	-3	-5	-7	-9
G	-4	-1	0	-2	-4	-4	-6
C	-6	-3	0	1	-1	-3	-5
T	-8	-5	-2	-1	2	0	-2
A	-10	-7	-4	-3	0	1	1

Table 3-2 Scoring Matrix.

	-	A	C	C	T	G	A
-	0	-2	-4	-6	-8	-10	-12
A	-2	1	-1	-3	-5	-7	-9
G	-4	-1	0	-2	-4	-4	-6
C	-6	-3	0	1	-1	-3	-5
T	-8	-5	-2	-1	2	0	-2
A	-10	-7	-4	-3	0	1	1

Table 3-3 Traceback Step.

And according to traceback step the best alignment for the two sequences will be:

A C C T G A

A G C T - A

3.8 Local Alignment (Smith-Waterman Algorithm):

A modification of the dynamic programming algorithm for sequence alignment provides a local sequence alignment giving the highest scoring local match between two sequences [33]. Local alignments are usually more meaningful than global matches because they include patterns that are conserved in the sequences. They can also be used instead of the Needleman-Wunsch algorithm to match two sequences that may have a matched region, that is only a fraction of their lengths, that have different lengths, that overlap or where one sequence is a fragment or subsequence of the other. The rules for calculating scoring matrix values are slightly different the most important differences being (1) the scoring system must include negative scores for mismatches and (2) when a dynamic programming scoring matrix value becomes negative that value is set to zero which has the effect

of terminating any alignment up to that point. The alignments are produced by starting at the highest scoring positions in the scoring matrix and following a Trace path from those positions up to a box that scores zero.

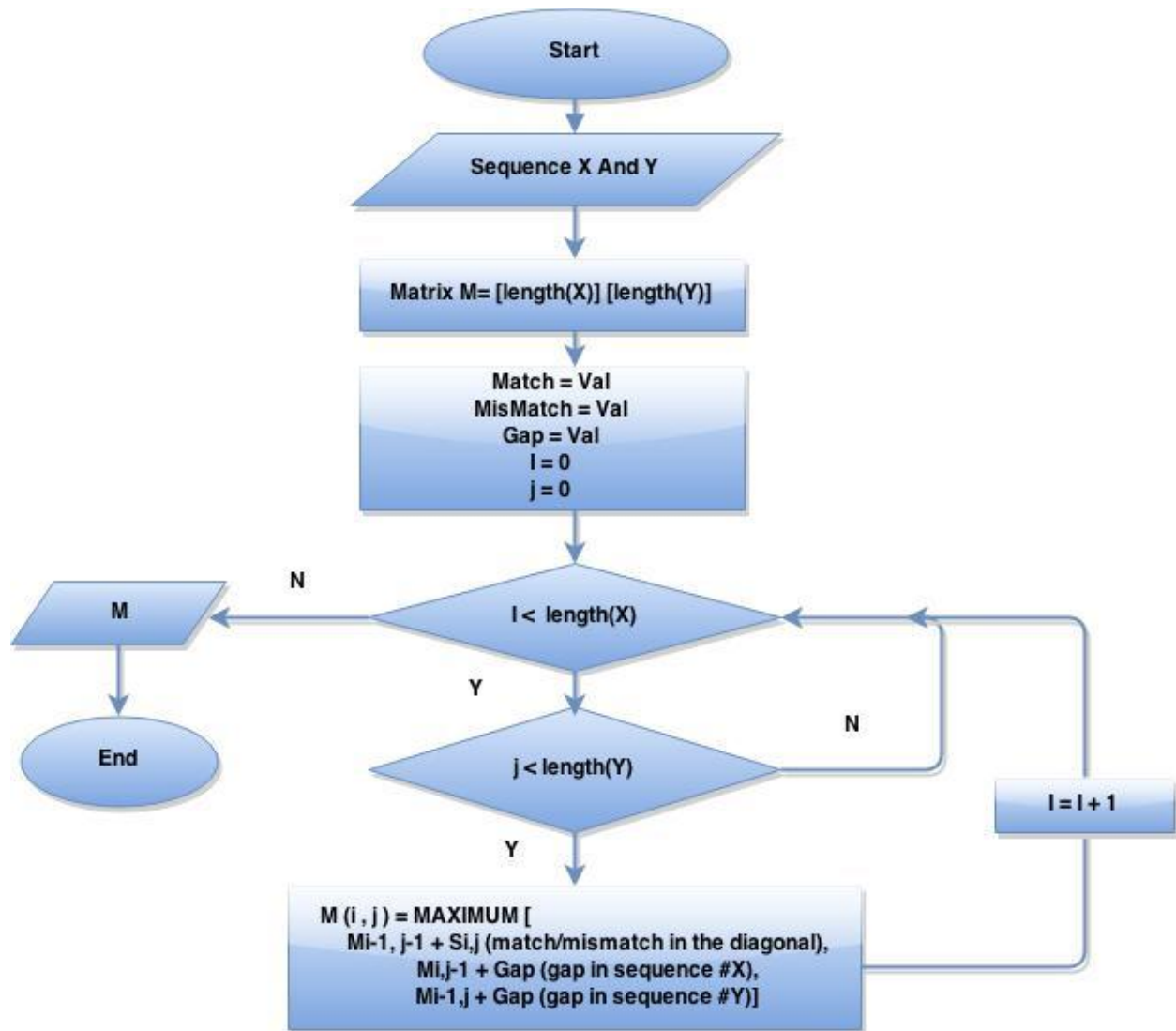
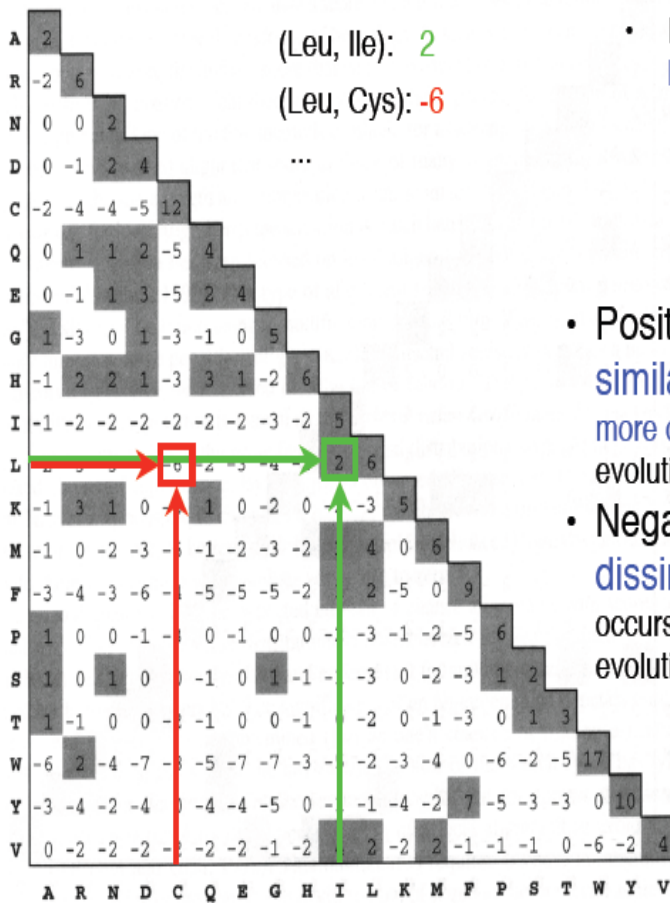


Figure 3.3 Sequence Alignment Flowchart.

3.9 Substitution matrices:

- In proteins some mismatches are more acceptable than others.
- Substitution matrices give a score for each substitution of one amino acid by another.



(Leu, Ile): 2
(Leu, Cys): -6
...

- For a set of well known proteins:
 - Align the sequences
 - Count the mutations at each position
 - For each substitution set the score to the log-odd ratio

$$\log\left(\frac{\textit{observed}}{\textit{expected by chance}}\right)$$

- Positive score: the amino acids are **similar**, mutations from one into the other occur **more often** than expected by chance during evolution
- Negative score: the amino acids are **dissimilar**, the mutation from one into the other occurs **less often** than expected by chance during evolution

PAM250

From: A. D. Baxeavanis, "Bioinformatics"

Figure 3.4 Example Of Substitution Matrix [34].

3.9.1 Different kind Of Matrices:

➤ **PAM series :**

Percent Accepted Mutation (Dayhoff M., 1968, 1972, 1978)[34]. A unit introduced by Dayhoff et al [34]. To quantify the amount of evolutionary change in a protein sequence. 1.0 PAM unit is the amount of evolution which will change on average 1% of amino acids in a protein sequence. A PAM(x) substitution matrix is a look-up table in which scores for each amino acid substitution have been calculated based on the frequency of that substitution in closely related proteins that have experienced a certain amount (x) of evolutionary divergence:

- Based on 1572 protein sequences from 71 families.
- Old standard matrix: PAM250.

➤ **BLOSUM series:**

Blocks Substitution Matrix (Henikoff S. & Henikoff JG., PNAS, 1992) [34]. A substitution matrix in which scores for each position are derived from observations of the frequencies of substitutions in blocks of local alignments in related proteins. Each matrix is tailored to a particular evolutionary distance. In the BLOSUM62 matrix, for example the alignment from which scores was derived were created using sequences sharing no more than 62% identity. Sequences more identical than 62% are represented by a single sequence in the alignment so as to avoid over-weighting closely related family members [34]:

- Based on alignments in the BLOCKS database.
- Standard matrix: BLOSUM62.

3.9.2 Limitations:

- Substitution matrices do not take into account long range interactions between residues.
- They assume that identical residues are equal (whereas in real life a residue at the active site has other evolutionary constraints than the same residue outside of the active site).
- They assume evolution rate to be constant.

3.10 Objective:

Implement Qt/C++ class for Similarity measure.

3.11 Material:

- C++ compiler gnu GCC for Unix/mac or VC++ for MS windows
- Qt SDK
- Any Computer platforms (Pc/Mac or UNIX).

3.12 Methods:

- Dynamic programming approach:
 - Global alignment (Needleman-Wunsch algorithm).
 - Local alignment (Smith-Waterman algorithm).

4 - Chapter Four

Case Study and Results

4.1 Microsatellite Repeats Case Study:

Apply FindMicrosatelliteRepeats on two types of Trinucleotide repeat disorder such as FXN gene with NCBI Reference Sequence: NM_181425.2 [35] and FMR1 (fragile X mental retardation 1) with NCBI Reference Sequence: L19493.1 [36]:

Motif	Start Position	Frequency	Length	MS Sequence
TA	9	3	2	TATATA
TT	333	3	2	TTTTTT
TT	340	3	2	TTTTTT
AT	547	3	2	ATATAT
TG	687	4	2	TGTGTGTG
AA	721	3	2	AAAAAA
TT	745	3	2	TTTTTT
TT	883	3	2	TTTTTT
TTGTTT	931	3	6	TTGTTTTTGTTTTTGTTT
GA	981	4	2	GAGAGAGA
GA	1037	3	2	GAGAGA

TT	1138	5	2	TTTTTTTTTT
GA	1207	3	2	GAGAGA
AT	1312	3	2	ATATAT
TT	1373	4	2	TTTTTTTT
TT	1382	3	2	TTTTTT
TT	1395	3	2	TTTTTT
TT	1423	3	2	TTTTTT
TT	1434	5	2	TTTTTTTTTT
CA	1782	3	2	CACACA
AT	1938	3	2	ATATAT
TG	2339	3	2	TGTGTG

Table 4.1 FRM1 Microsatellite repeats Search Result.

Motif	Start Position	Frequency	Length	MS Sequence
CCCAG	265	3	5	CCCAGCCCAGCCCAG
CT	483	3	2	CTCTCT
TT	544	3	2	TTTTTT
TT	934	3	2	TTTTTT
GTT	940	5	3	GTTGTTGTTGTTGTT
TT	957	4	2	TTTTTTTT
TT	1070	3	2	TTTTTT
AA	1157	3	2	AAAAAA
TG	1499	3	2	TGTGTG
AT	1506	4	2	ATATATAT

AA	1560	3	2	AAAAAA
AA	1589	3	2	AAAAAA
AG	1707	3	2	AGAGAG
AG	1808	3	2	GAGAGA
TA	1924	3	2	TATATA
AA	2128	3	2	AAAAAA
AAT	2284	3	3	AATAATAAT
ATA	2299	5	3	ATAATAATAATA
AA	2618	7	2	AAAAAAAAAAAAA
AA	2834	3	2	AAAAAA
AA	2842	4	2	AAAAAAAA
TAA	2856	3	3	TAATAATAA
TT	2898	3	2	TTTTTT
TT	2918	3	2	TTTTTT
AA	3403	3	2	AAAAAA
GG	3483	3	2	GGGGGG
AAAT	3558	5	4	AAATAAATAAATAAAT
TG	3658	3	2	TGTGTG
TT	3823	4	2	TTTTTTTT
TT	3841	3	2	TTTTTT
TT	3978	3	2	TTTTTT
GT	4081	3	2	GTGTGT
CT	4191	3	2	CTCTCT
GT	4219	3	2	GTGTGT
TT	4316	7	2	TTTTTTTTTTTTTT

TG	4765	3	2	TGTGTG
AC	4821	3	2	ACACAC
TT	5025	3	2	TTTTTT
AA	6027	3	2	AAAAAA
TA	6078	3	2	TATATA
TC	6548	3	2	TCTCTC
GG	6853	3	2	GGGGGG
CT	6901	3	3	CTCCTCCTC
CA	7086	3	2	CACACA

Table 4.2 FXN Microsatellite repeats Search Result.

BioQt Microsatellite repeats finder

Length of repeated sequence:
 Minimum: Maximum:
 Minimum number of repeats:
 Minimum length of tandem repeat:

Search Result

motif	postion	frequency	Length	Sequence
AA	3403	3	2	AAAAAA
GG	3483	3	2	GGGGGG
AAAT	3558	5	4	AAATAAATAAATAAATAAT
TG	3658	3	2	TGTGTG

Figure 4.1 BioQt Microsatellite repeats finder.

4.2 Palindromes Case Study:

- Test mitochondrial genome with telomeres of *Polytomella magna* mitochondrion with GenBank accession KC733827 [37] against both BioQt palindrome Algorithms.
- Test the Dummy sequence against both BioQt palindrome Algorithms :

```
AACAATGCCATGATGATGATTATTACGACACAACAACACCGCGCTTGA  
CGGCGGCGGATGGATGCCGCGATCAGACGTTCAACGCCACGTAACGT  
AACGCAACGTAACCTAACGACACTGTTAACGGTACGAT
```

4.2.1 Results:

4.2.1.1 Experiment 1:

- dynamic programming approach :

Test KC733827 sequence against BioQt dynamic programming Palindrome finder algorithm with minimum palindrome length 10 and maximum length 20:

Palindromic sequence	Position
AGTCCGGACT	617
GGATCGATCC	708
ACTTTTAAAGT	1021
ATTCTCAGAAT	1100
TGAGTACTCA	1119
TTTTAATTTAAAA	1729
AAGCTAGCTT	2703
CATGTTTAAACATG	2720

GTTAATCATTA	2762
TTTTAATTTAAAA	4320

Table 4.3 KC733827 Palindromes.

- Naïve Algorithm: Crashed down.

4.2.1.2 Experiment 2:

- dynamic programming approach :

Test dummy sequence against BioQt dynamic programming Palindrome finder algorithm with minimum palindrome length 6 and maximum length 20:

Palindromic sequence	Position
ATGCCAT	4
CATGATG	8
CGTTCAACG	75
ACGTAACGT	87
CGTAACG	93
ACACTGT	115
GTTAAC	120

Table 4.4 Dummy Sequence Palindromes.

- Naive Algorithm :
Only one palindrome detected at position 120 → GTTAAC.

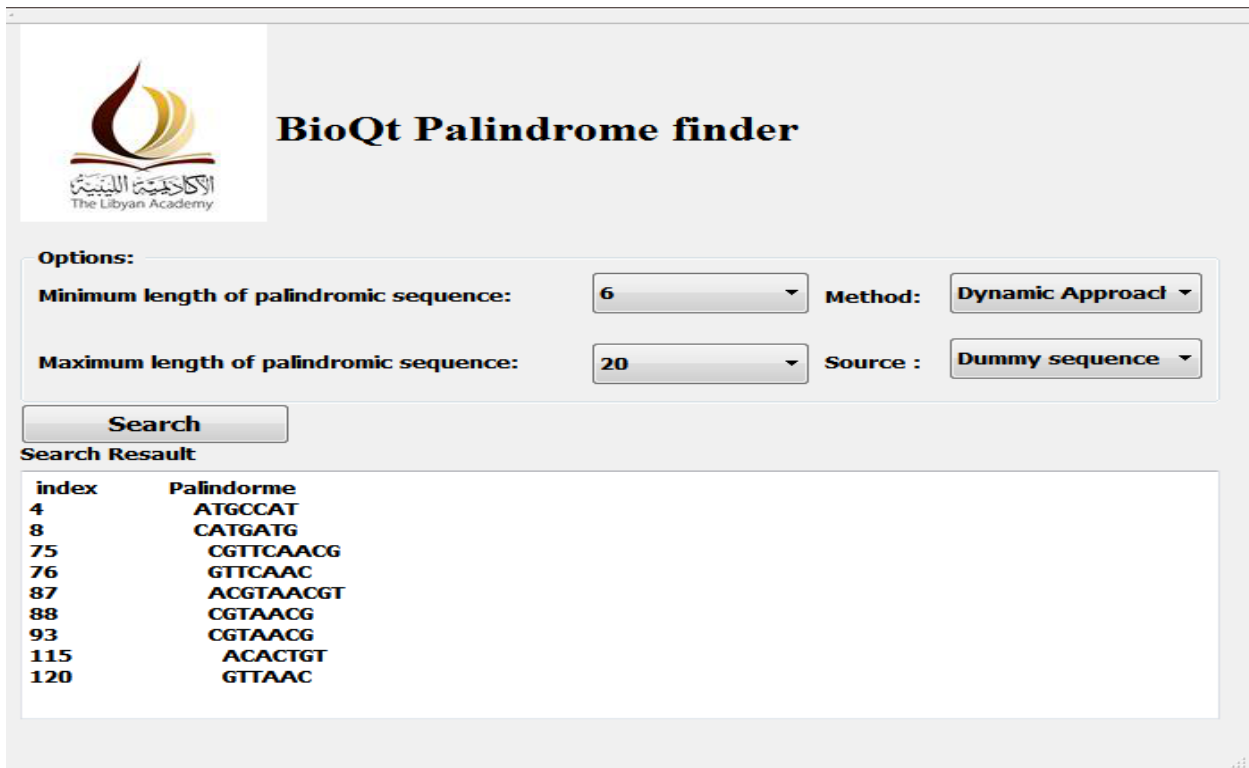


Figure 4.2 Palindrome Finder Dynamic Approach.

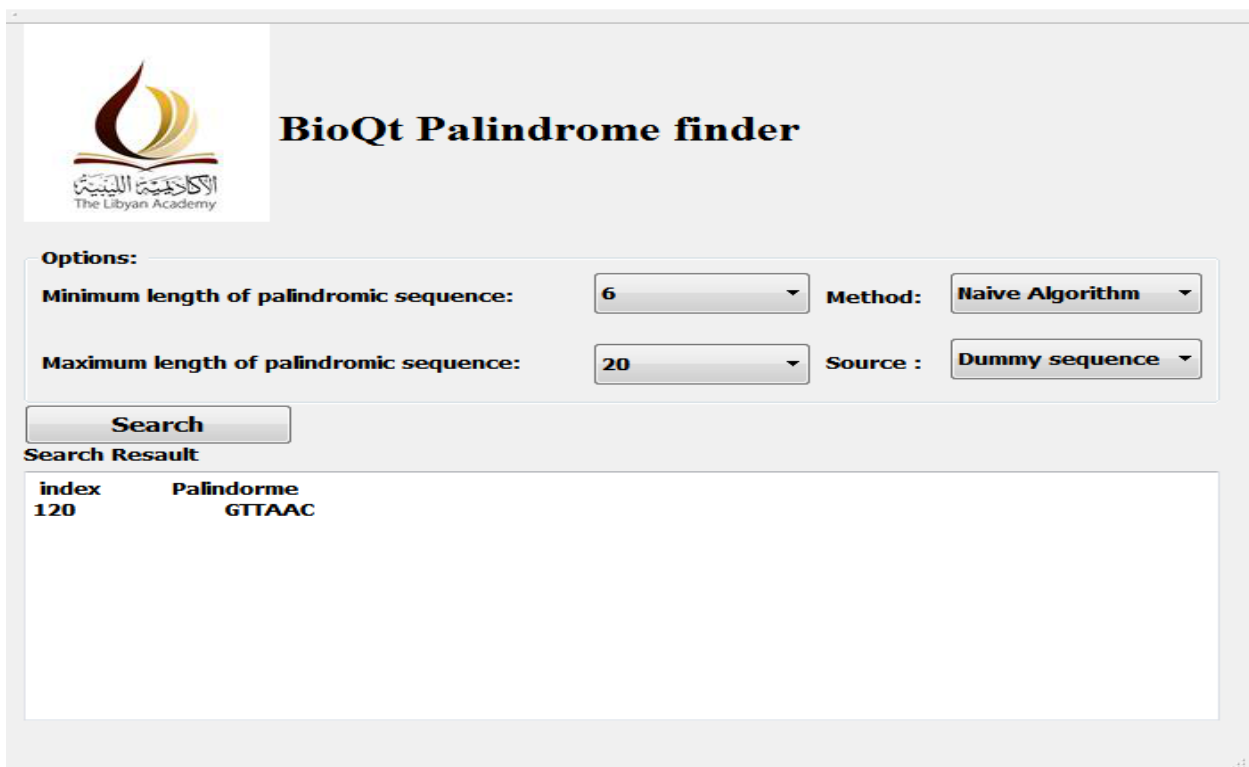


Figure 4.3 Palindrome Finder Naive Algorithm.

4.3 Exact String Matching:

Test mitochondrial genome with telomeres of *Polytomella magna* mitochondrion with GenBank accession KC733827 [37] against Knuth–Morris–Pratt algorithm(used by BioQt for exact string matching) and compare the execution time with other exact string matching algorithms Brute Force , Boyer Moor and Shift-or .

4.3.1 Experiment:

Using the sequence TGAGAT as pattern.

4.3.2 Results:

All algorithms give the same result:

The pattern TGAGAT matched at text indices 1644, 1849, 2219, 4235, 4440 and 4810.

Name of algorithms	Brute Force	Boyer Moor	Knuth–Morris–Pratt	Shift-or
Numbers of running time				
1	795995	364783	344048	874327
2	804442	516072	324848	802906
3	774492	513000	523367	529895
4	769500	336752	496105	809818
5	773724	334832	491881	772956
6	777180	513768	323313	828249
7	773724	540647	329456	803674
8	767580	342128	322929	756445
9	796763	556006	337520	761052
10	803290	342896	338288	773724

Table 4.5 Exact String Matching.

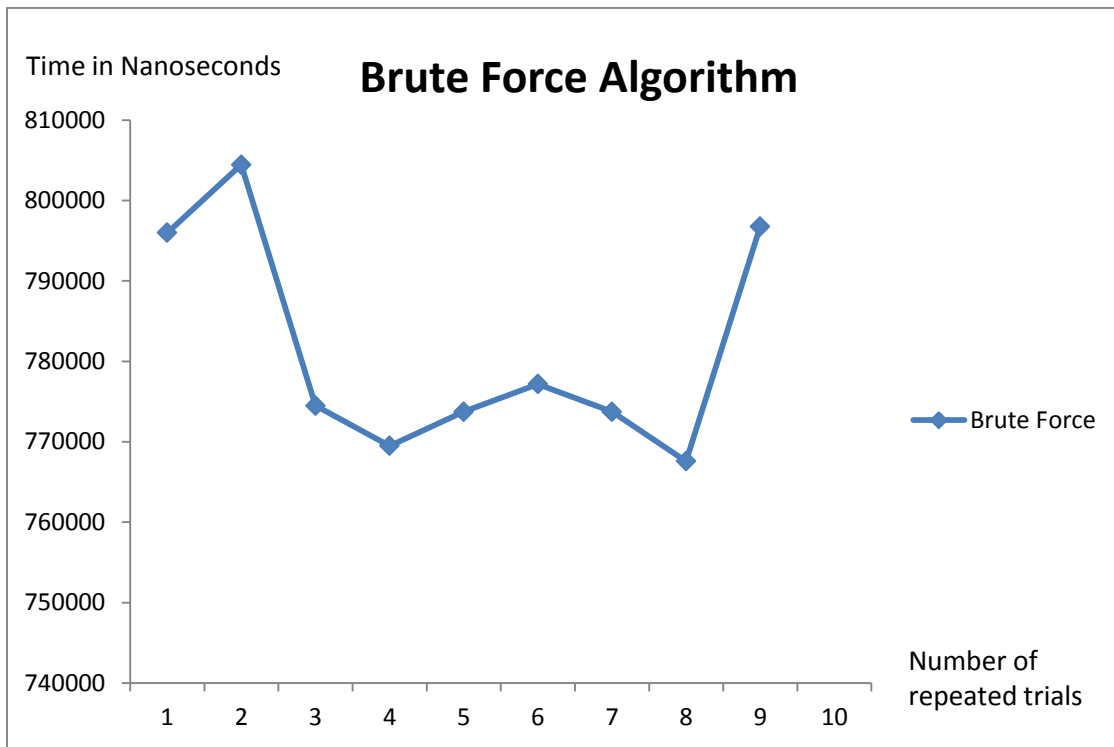


Figure 4.5 Brute Force Algorithm running time.

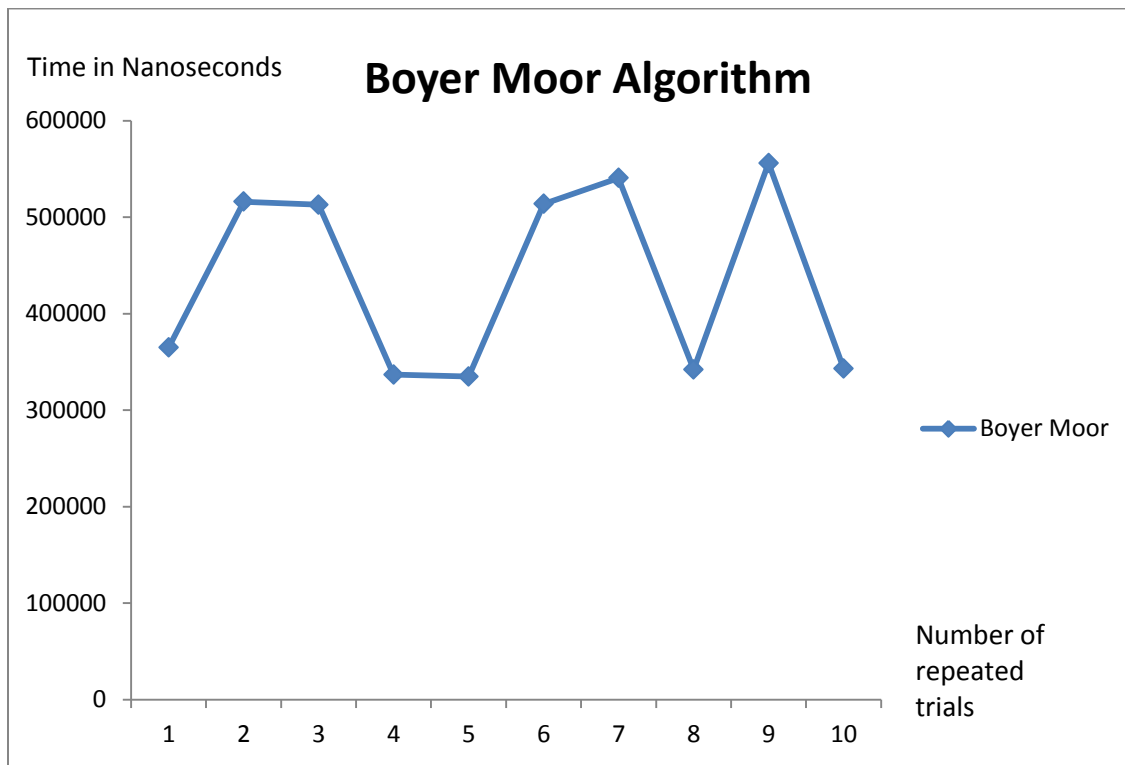


Figure 4.6 Boyer Moor Algorithm running time.

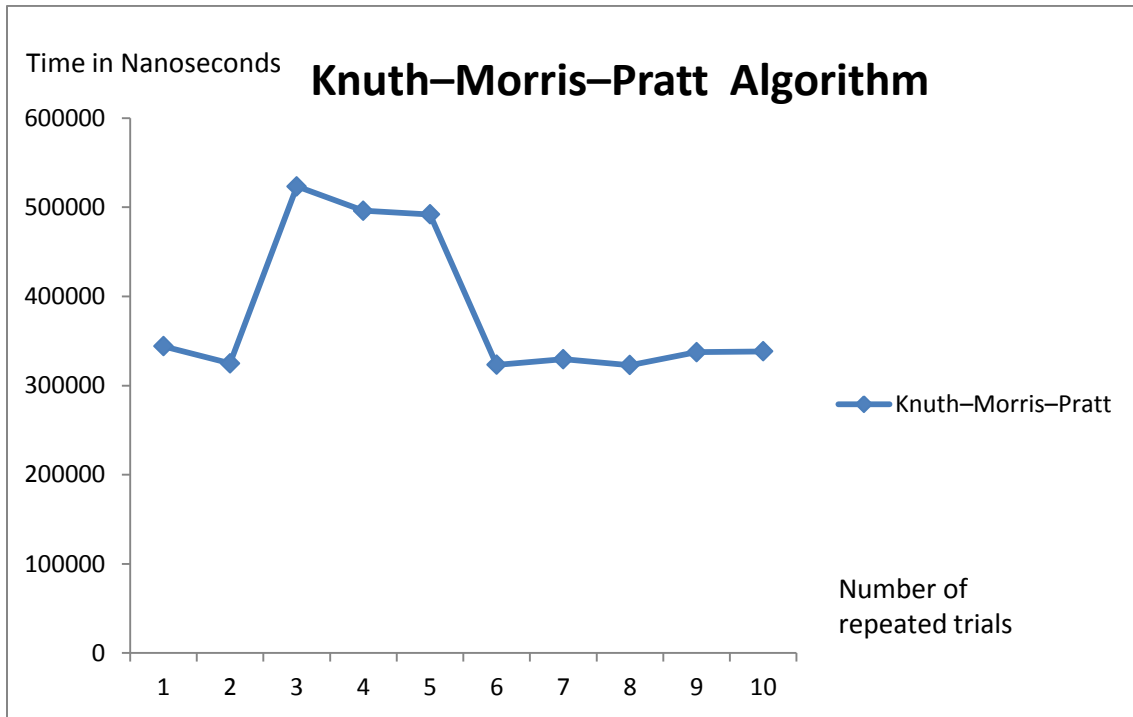


Figure 4.7 Knuth-Morris Algorithm running time.

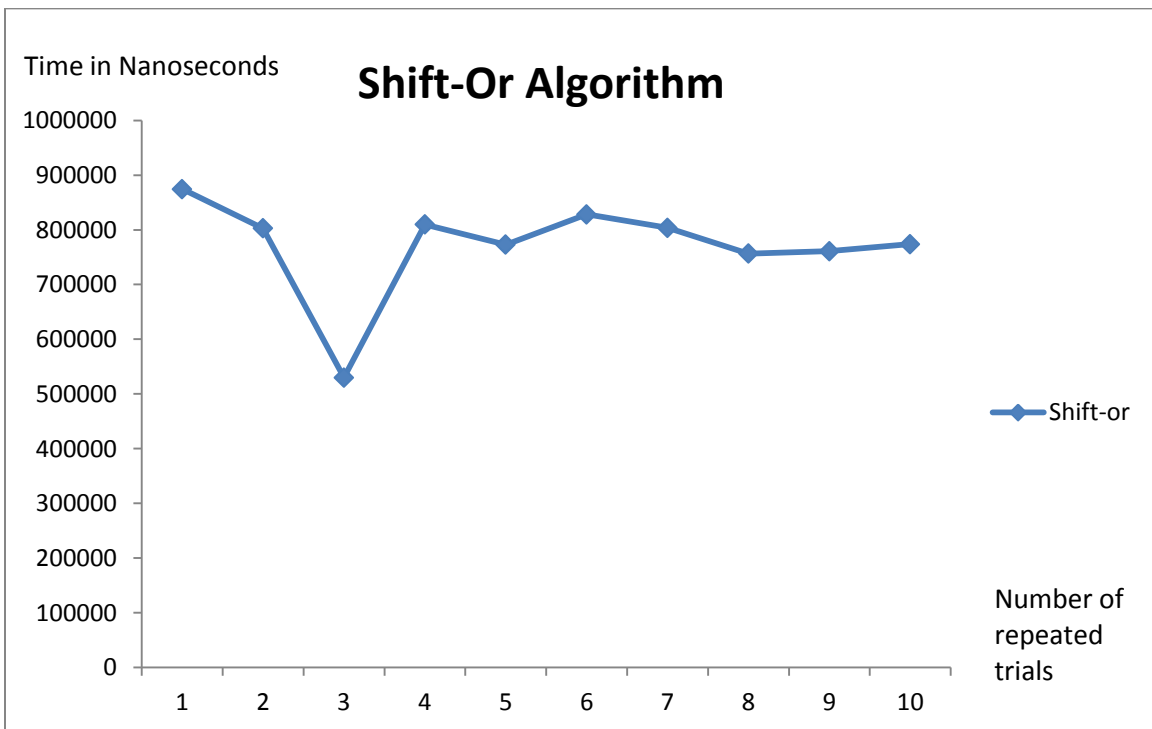


Figure 4.8 Shifts-Or Algorithm running time.

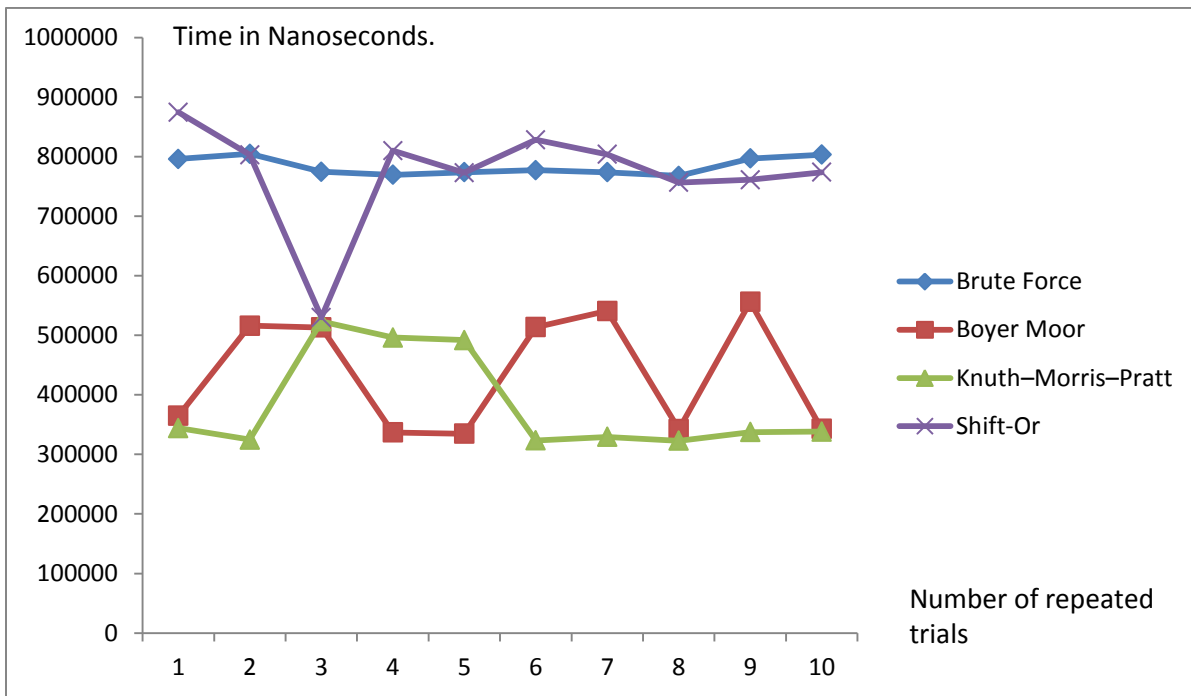


Figure 4.9 String Matching Algorithm running time.

BioQt Exact String Matching

Text: GGAGGGGTGGAGTCGTGACGTGGGCAAGCTGACCCTGAAGTTCATCTGCACCACCGGCAAGCTG
 CCGTGCCCTGGCCACCCTCGTGACCACCCTGACCTACGGCGTGCAGTGCTTCAGCCGCTACCCC
 GACCACATGAAGCAGCACGACTTCTTCAAGTCCGCCATGCCCGAAGGCTACGTCCAGGAGCGC
 ACCATCTTCTCAAGGACGACGGCAACTACAAGACCCGCGCCGAGGTGAAGTTCGAGGGCGAC
 ACCCTGGTGAACCGCATCGAGCTGAAGGGCATCGACTTCAAGGAGGACGGCAACATCCTG666
 CACAAGCTGGAGTACAACACAACAGCCACAACGTCTATATCATGGCCGACAAGCAGAAGA
 ACCGATCAAGGTGAAGTTCAGATTCGGCCACAACATCGAGGACGGCAAGGCTGACGCTGGCC

Pattern: TGAGAT

Search Algorithm: **KMP** Samples: **Sample**

Results: Execution Time In Nano Second: **356719**

1644
 1849
 2219
 4235
 4440
 4810

Figure 4.10 KMP Algorithm Output.

4.4 Sequence Alignment:

Test two dummy sequences against Sequence Alignment Algorithms:

- Hirschberg's Algorithm (LCS).
- Needleman–Wunsch algorithm
- Smith-Waterman Algorithm

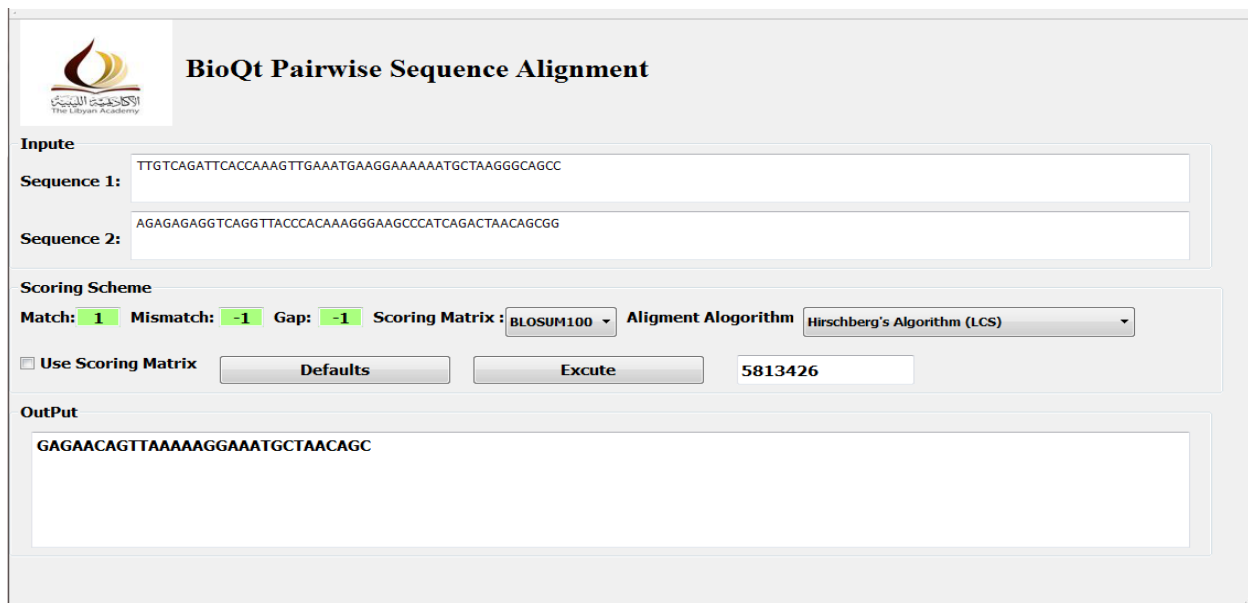
And compare the execution time.

4.4.1 Experiment:

Using two dummy sequences:

```
s1=TTGTCAGATTCACCAAAGTTGAAATGAAGGAAAAAATGCTAAGGGCAGCC
s2=AGAGAGAGGTCAGGTTACCCACAAAGGGAAGCCCATCAGACTAACAGCGG
```

4.4.2 Results:



BioQt Pairwise Sequence Alignment

Input

Sequence 1: TTGTCAGATTCACCAAAGTTGAAATGAAGGAAAAAATGCTAAGGGCAGCC

Sequence 2: AGAGAGAGGTCAGGTTACCCACAAAGGGAAGCCCATCAGACTAACAGCGG

Scoring Scheme

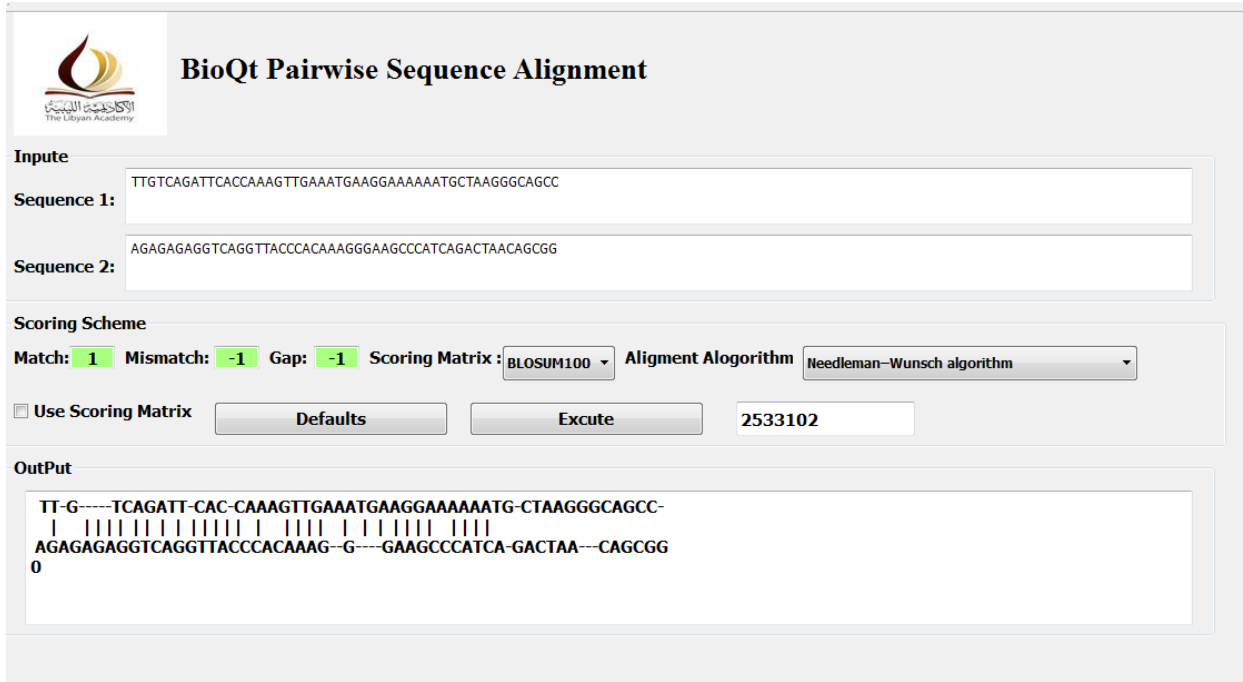
Match: 1 Mismatch: -1 Gap: -1 Scoring Matrix: BLOSUM100 Alignment Algorithm: Hirschberg's Algorithm (LCS)

Use Scoring Matrix Defaults Excute 5813426

OutPut

GAGAACAGTTAAAAGGAAATGCTAACAGC

Figure 4.12 Hirschberg's Algorithm Output alignment.



BioQt Pairwise Sequence Alignment

Inpute

Sequence 1: TTGTCAGATTACCAAAGTTGAAATGAAGGAAAAAATGCTAAGGGCAGCC

Sequence 2: AGAGAGAGGTCAGGTTACCCACAAAGGGAAGCCCATCAGACTAACAGCGG

Scoring Scheme

Match: 1 Mismatch: -1 Gap: -1 Scoring Matrix: BLOSUM100 Alignment Alogorithm: Needleman-Wunsch algorithm

Use Scoring Matrix Defaults Excute 2533102

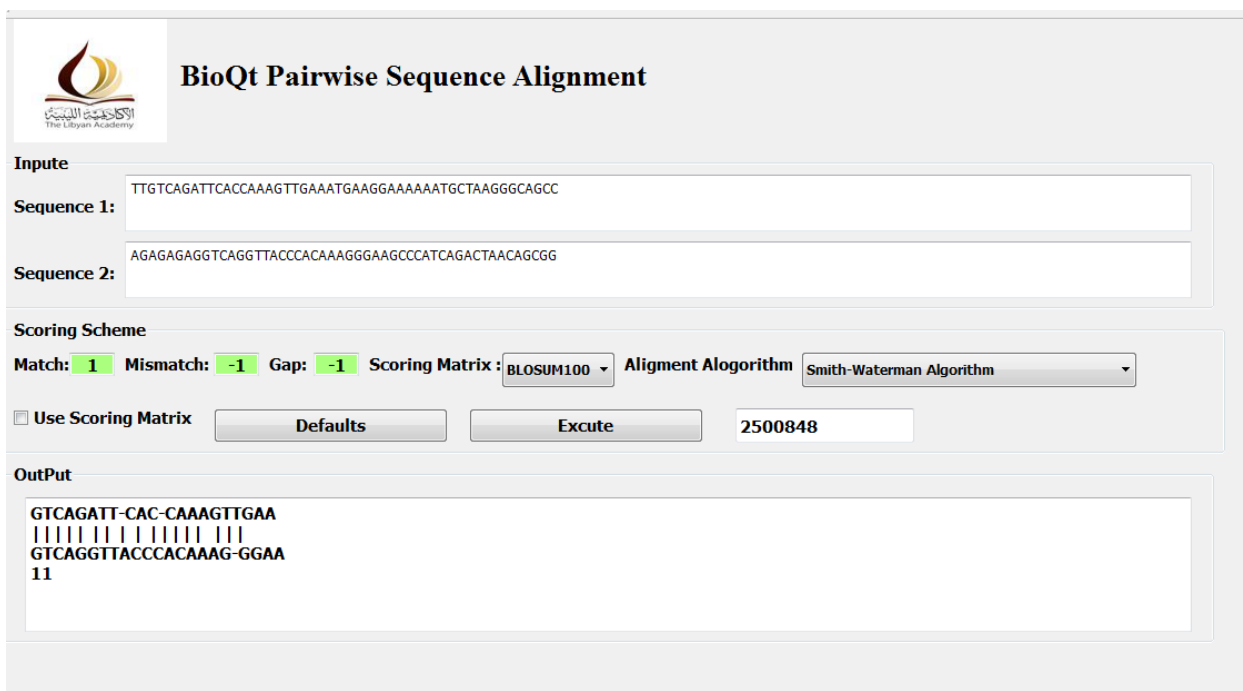
OutPut

```

TT-G-----TCAGATT-CAC-CAAAGTTGAAATGAAGGAAAAAATG-CTAAGGGCAGCC-
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
AGAGAGAGGTCAGGTTACCCACAAAG-G---GAAGCCCATCA-GACTAA---CAGCGG
0

```

Figure 4.13 Needleman-Wunsch algorithm Output alignment.



BioQt Pairwise Sequence Alignment

Inpute

Sequence 1: TTGTCAGATTACCAAAGTTGAAATGAAGGAAAAAATGCTAAGGGCAGCC

Sequence 2: AGAGAGAGGTCAGGTTACCCACAAAGGGAAGCCCATCAGACTAACAGCGG

Scoring Scheme

Match: 1 Mismatch: -1 Gap: -1 Scoring Matrix: BLOSUM100 Alignment Alogorithm: Smith-Waterman Algorithm

Use Scoring Matrix Defaults Excute 2500848

OutPut

```

GTCAGATT-CAC-CAAAGTTGAA
| | | | | | | | | | | | | | | |
GTCAGGTTACCCACAAAG-GGAA
11

```

Figure 4.14 Smith-Waterman Algorithm Output alignment.

Name of algorithms	Hirschberg's Algorithm (LCS)	Needleman– Wunsch algorithm	Smith-Waterman Algorithm
Numbers of running time			
1	7179236	2366456	2314619
2	5582271	1955983	2316923
3	5668282	1937936	2277757
4	5566528	1892627	3778727
5	8565780	1878420	4030232
6	9563355	1899923	3731114
7	5701689	1933329	2354936
8	5696313	1876116	2305787
9	5600318	1945616	2310395
10	5661371	1891091	2275069

Table 4.6 Sequence Alignment.

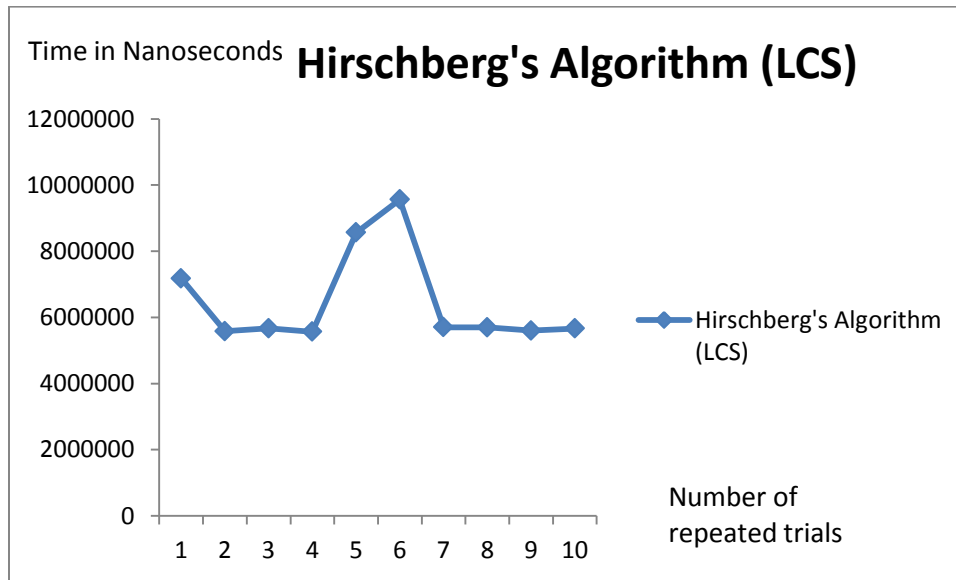


Figure 4.15 Hirschberg's Algorithm running time.

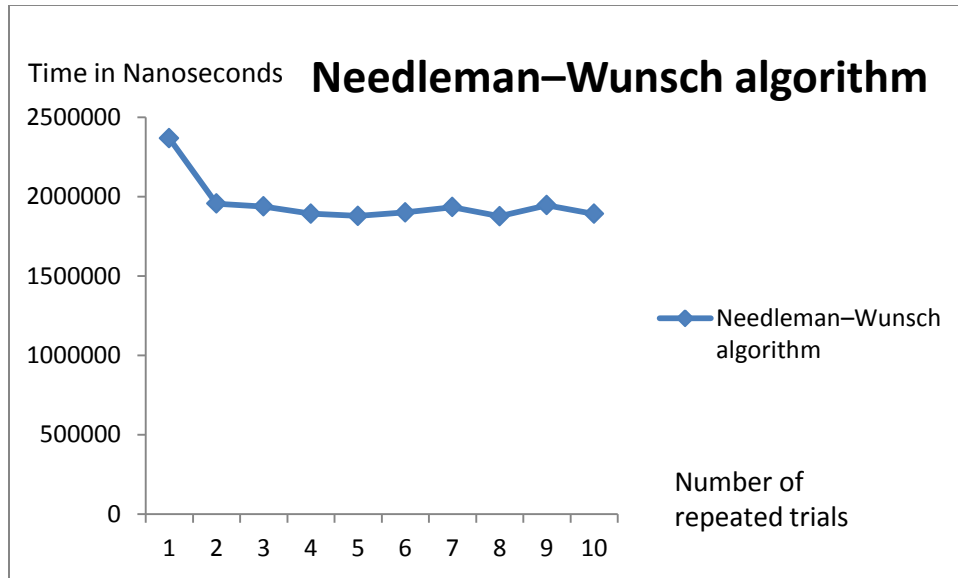


Figure 4.16 Needleman–Wunsch algorithm running time.

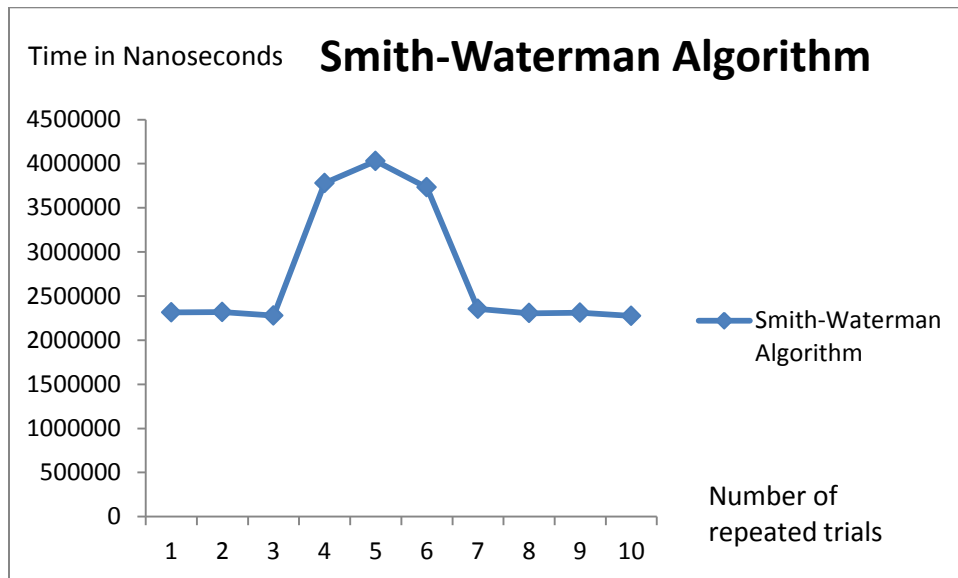


Figure 4.17 Smith-Waterman Algorithm running time.

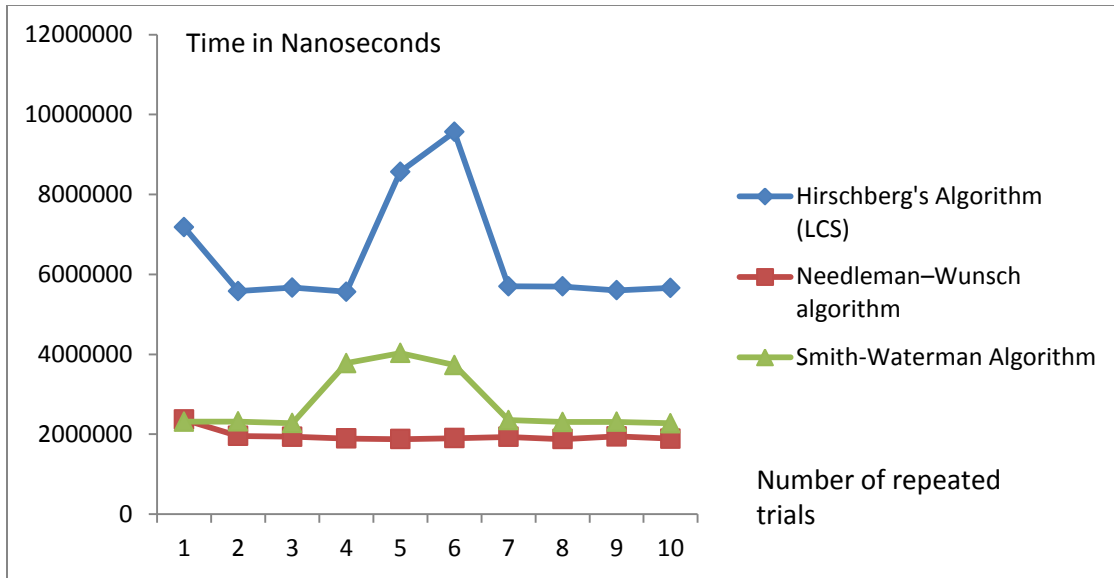


Figure 4.18 comparison between alignment algorithms running time.

5 - Chapter Five:

Conclusion and Future work

In this thesis the evaluation of code implementation has been successfully done for all algorithms: exact string matching problem, Microsatellite Repeats, Palindromic sequences, longest common subsequence and Needleman-Wunsch and Smith-Waterman sequence alignment. . In this implementation we use C++ language and the Qt SDK 4.8 with the GCC 4.8 compiler to test the algorithms under the Centos Linux version 6 Operating system with RAM 8GB. Evaluating the same sequences on different algorithms may show different results. While some of the differences are shown to be expected and are part of the different default considerations or interpretations of those algorithms. Implementation of dynamic programming algorithm to compare sequences shows remarkable waste of time compared to BLAST or FASTA (this is accepted due to nature of algorithms). Furthermore for future work complete the classes deal with open reading frames, restriction enzyme, PCR primer design and database file handler.

6 - References:

- 1- Margaret O. Dayhoff. Atlas of Protein Sequence and Structure: Supplement, Volume 3. National Biomedical Research Foundation.(1978).
- 2- Neil C.Jones and Pavel A.Pevzner. An introduction to:bioinformatics algorithms. The MIT Press.(2004).
- 3- Dinesh Bhatia.Medical Informatics.PHI Learning Pvt. Ltd.(2015).
- 4- Jasmin Blanchette; Mark Summerfield. C++ GUI Programming with Qt 4, Second Edition. Prentice Hall. (2008).
- 5- The Qt Company and ESA to develop 3D mapping and visualization software to support the Rosetta science operations. December 18, 2014. Helsinki, Finland. <https://www.qt.io/qt-news/qt-company-esa-develop-3d-mapping-visualization-software-support-rosetta-science-operations/>.
- 6- DreamWorks Animation keynote at Qt Developer Days. September 7th, 2010. <https://blog.qt.io/blog/2010/09/07/dreamworks-animation-keynote-at-qt-developer-days/>.
- 7- Peter Kreußel .Lucasfilm Goes for Trolltech's Qt. Oct 16, 2007. <http://www.linux-magazine.com/Online/News/Lucasfilm-Goes-for-Trolltech-s-Qt>.
- 8- Gary Townsend. Panasonic Avionics Inflight Entertainment. <https://mar-eu-1-pa6s6zn8.qtcloudapp.com/case-panasonic/>.
- 9- Qt on Android. <https://www.qt.io/case-adeneo-embedded/>.
- 10- David L.Nelson and Michael M.Cox. Lehninger Principles of Biochemistry, sixth edition. Freeman, W. H. & Company.(2012).
- 11- Barbara Hansen and Lynn Jorde. USMLE Step 1 - Biochemistry and Medical Genetics Lecture Notes. Kaplan.(2013).
- 12- Mona Snigh .Topics in computational molecular biology. lecture2. (1999).

- 13- Ben Langmead. Strings and exact matching. Johns Hopkins University Whiting School of Engineering Computer Science.(2014).
- 14- Litt, M.; Luty, J.A. A hypervariable microsatellite revealed by in vitro amplification of a dinucleotide repeat within the cardiac muscle actin gene. *Am. J. Hum. Genet.* (1989).
- 15- Peter D. Turnpenny and Sian Ellard. *Emery's Elements of Medical Genetics*, 13th. ed. Elsevier.(2007).
- 16- Gous Miah, Mohd Y. Rafii et al. A Review of Microsatellite Markers and Their Applications in Rice Breeding Programs to Improve Blast Disease Resistance. *Int. J. Mol. Sci.* 2013, 14, 22499-22528; doi:10.3390/ijms141122499.
- 17- Hans Ellegren. Microsatellites: simple sequences with complex evolution. *Nature Reviews Genetics* 5, 435-445 (June 2004) | doi:10.1038/nrg1348.
- 18- E. Yu. Siyanova and S. M. Mirkin. Expansion of Trinucleotide Repeats. Department of Molecular Genetics, University of Illinois at Chicago, Chicago, IL 60607, USA. (October 2, 2000).
- 19- Duncan FISHWICK, M.A. An Early Christian Cryptogram. University of St. Michael's College, Toronto. *CCHA, Report*, 26 (1959), 29-41.
- 20- David Roy Smith et al. Palindromic Genes in the Linear Mitochondrial Genome of the Nonphotosynthetic Green Alga *Polytomella magna*. *Genome Biol Evol.* 2013; 5(9): 1661–1667. [PMCID: PMC3787674].
- 21- Horace R. Drew, Denise Lewy, Jason Conaty, Keith N. Rand, Philip Hendry and Trevor Lockett. RNA hairpin loops repress protein synthesis more strongly than hammerhead ribozymes. CSIRO Division of Molecular Science, North Ryde, Australia. *Eur. J. Biochem.* 266, 260±273 (1999).
- 22- Choi and C.Q. DNA palindromes found in cancer. *Genome Biology*.(2005).

- 23- Hisashi Tanaka, Donald A. Bergstrom, Meng-Chao Yao and Stephen J. Tapscott. Widespread and nonrandom distribution of DNA palindromes in cancer cells provides a structural platform for subsequent gene amplification. *Nature Genetics* 37, 320 - 327 (2005) .
- 24- Lisnic B, Svetec IK, Saric H, Nikolic I and Zgaga Z. Palindrome content of the yeast *Saccharomyces cerevisiae* genome. *Curr Genet.* 2005 May; 47(5):289-97. Epub 2005 Mar 18.
- 25- D.E. Knuth, J.H. Morris, Jr., and V.R. Pratt: Fast pattern matching in strings. *SIAM Journal on Computing.* 323–350(1977).
- 26- R.S. Boyer, J.S. Moore, A fast string searching algorithm, *Communication of the ACM*, Vol. 20, No. 10.(1977).
- 27- Ben Langmead. Boyer-Moore. john hopkins whiting school of engineering. [*engineering.jhu.edu*].
- 28- Gibbs, A. J. and McIntyre, G. A. (1970) *European J. Biochem.* **16**, 1-11.
- 29- David Mount. *Bioinformatics: Sequence and Genome Analysis*, Second Edition.(2004). Cold Spring Harbor Laboratory Press.
- 30- *Bioinformatics ALGORITHMS: Techniques and Applications*.(2008). Ion I. Mandoiu & Alexander Zelikovsky. John Wiley & Sons, Inc.
- 31- C. S. Needleman, C. D. Wunsch. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, vol. 48, no. 1, pp. 443-453.
- 32- Smith T.F. & Waterman M.S. (1981). Identification of common molecular subsequences *Journal of molecular biology.* 147: 195–197.
- 33- Smith, Temple F and Waterman, Michael S. (1981) “Identification of Common Molecular Sub sequences”. *Journal of Molecular Biology* 147: 195–197.

- 34- Pairwise sequence alignments; Volker Flegel, Vassilios Ioannidis; VI - 2004; <http://www.ch.embnet.org/CoursEMBnet/Zurich04/slides/pairwise.pdf>.
- 35- <http://www.ncbi.nlm.nih.gov/nucore/239787185?report=genbank>.
- 36- <http://www.ncbi.nlm.nih.gov/nucore/L19493.1>.
- 37- <http://www.ncbi.nlm.nih.gov/nucore/KC733827>.