Efficient Pattern Matching With Flexible Wildcard Gaps and One-off Constraint

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Submitted By
Anu Dahiya
(Roll No. 801232004)

Under the supervision of:

Dr. Deepak Garg
Associate Professor, CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2014

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Efficient Pattern Matching with Flexible Wildcard Gaps and One-off Constraint*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Deepak Garg* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

(Anu Dahiya)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Deepak Garg)
Associate Professor,
CSED
Thapar University
Patiala

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

ACKNOWLEDGEMENT

I am grateful to many people for believing in me and my idea and thus giving me the encouragement and support in making this publication successful.

My deepest thanks to *Dr. Deepak Garg*, head of Computer Science and Engineering Department, for not only his expertise and guidance but also for helping flesh out the initial idea. His clear thinking aided to bring out this research in a concise and understandable manner.

I also owe my deepest gratitude to all the staff members of Computer Science and Engineering Department for providing the resources which kept this project on the fast track.

I would also like to thank my friends who stayed by my side and gave me encouragement to stay authentic in my research and assisted me in keeping a positive outlook while preparing this thesis.

Lastly, I would like to count my blessings for being surrounded by a loving family that gave me the boundless support throughout this project.

Anu Dahiya

(801232004)

ABSTRACT

Deoxyribonucleic acid (DNA) is the storehouse of all information and genetic instructions used in the development and functioning of a cell. The amount of DNA that is being extracted from the organism is increasing at a faster rate. With the considerable increase in the amount of biosequence data, there is need to develop new methods to extract knowledge from the data. Pattern matching is a basic operation in finding knowledge from large amount of biosequence data. Finding patterns help in analyzing the property of a sequence. Analyzing the DNA sequence can help in identifying the genetic diseases.

Promoter and intron in a DNA sequence does not occur consecutively but with a gap of 30-50 characters between them. So Pattern matching with wildcards is of great significance in bioinformatics. This thesis focuses on the problem of maximal pattern matching with flexible wildcard gaps and length constraints under the one-off condition. The problem is to find the maximum number of occurrences of a pattern P with user specified wildcard gap between every two consecutive letters of P in a biological sequence S under the one-off condition and constraint on the overall length of the matching occurrence. To obtain the optimal solution for this problem is difficult. For this problem, no complete solution has been developed so far. All algorithms are based on greedy approaches.

In this work, different existing algorithms for solving the problem of maximal pattern matching with flexible wildcard gaps and length constraints under the one-off condition have been studied along with their merits and de-merits. These algorithms are then compared on the basis of data structure used by them, technique incorporated in the algorithm, time and space complexities.

A heuristic algorithm, MOGO, based on the Nettree data structure has been proposed to solve this problem. Theoretical analysis and experimental results demonstrate that this algorithm performs better than the existing algorithms in most of the cases when tested on real world biological sequences.

TABLE OF CONTENTS

Certificate .... **i**

Acknowledgement .... **ii**

Abstract .... **iii**

Table of Contents .... **iv**

List of Figures .... **vii**

List of Tables .... **ix**

1. Introduction **1**

1.1 Pattern Matching 1

1.2 DNA 3

1.3 Pattern Matching in DNA Sequences 5

1.4 Significance of DNA Sequence Analysis 5

1.5 Pattern Matching with Wildcard Characters 6

1.6 Constraints 7

2. Literature Survey **9**

2.1 Basic Concepts and Data Structures Used 9

2.1.1 Bit Parallelism 9

2.1.2 Trie 9

2.1.3 Automata 10

2.1.4 Suffix Tree.. 11

2.2 Different Matching Strategies 12

2.2.1 Prefix Searching 13

2.2.2 Suffix Searching 14

2.2.3 Factor Searching 14

2.3 Variations in Traditional Pattern Matching Problem 15

2.3.1 Fixed Wildcard Characters 15

2.3.2 Variable Wildcard Gap 15

2.3.3 One-off Condition 16

2.3.4 Approximate Pattern Matching 18

2.4 Comparison of various algorithms involving one-off condition 18

3. Problem Statement 20

2.1 Problem Definition 20

2.2 Gap Analysis 22

2.3 Proposed Objective 22

2.4 Methodology Used 23

4. Algorithm 24

4.1 Data Structure 24

4.2 Algorithm Description 29

4.3 Complexity Analysis 32

4.4 An Illustration Example 32

5. Implementation and Experimental Results 35

5.1 Configuration and Architecture 35

5.2 Snapshots 36

5.3 Experimental Results 40

5.3.1 Experiment on Real Data 40

5.3.2 Experiment on Artificial Data 42

6. Conclusion and Future Scope 45

6.1 Conclusion 45

6.2 Future Scope 46

References 47

List of Publications 52

LIST OF FIGURES

Figure 1.1 Structure of DNA 4

Figure 1.2 Central Paradigm of Bioinformatics 5

Figure 1.3 Example showing different cases of Pattern Matching 8

Figure 2.1 Trie for the set of strings, S={atatc, atcg, atcgac} 10

Figure 2.2 Deterministic Automata 10

Figure 2.3 Non-Deterministic Automata 11

Figure 2.4 Suffix tree for the string “banana” 12

Figure 2.5 Categorization of searching approaches 13

Figure 4.1 Structure of Nettree Node 24

Figure 4.2 Structure of Nettree Level 25

Figure 4.3 Step-by-Step Creation of Nettree 27

Figure 4.4 Final Nettree 28

Figure 4.5 A Nettree 32

Figure 4.6 Nettree with min_occ and max_occ of each node 33

Figure 4.7 Removal of nodes from Nettree 34

Figure 4.8 Updated Nettree 34

Figure 5.1 CGI Architecture 35

Figure 5.2 First Page 36

Figure 5.3 Input File containing the DNA sequence 37

Figure 5.4 Output when wrong pattern is entered by the user 37

Figure 5.5 Output when path of file is not given 38

Figure 5.6 Output when invalid range of global length constraints is provided..... 38

Figure 5.7 Output without one-off condition 39

Figure 5.8 Output with one-off condition 39

Figure 5.9 Effect of the length of pattern on the accuracy of MOGO… 43

Figure 5.10 Effect of maximum wildcard gap on the accuracy of MOGO 43

LIST OF TABLES

Table 2.1 Comparison of Algorithms 19

Table 3.1 Different alignments of Pattern with Sequence in Example 3 21

Table 4.1 An Instance of Number_levels Array 33

Table 5.1 Biological Sequences 40

Table 5.2 Patterns with wildcard gaps 40

Table 5.3 Global Length Constraints 41

Table 5.4 Experimental Results 41

CHAPTER – 1
INTRODUCTION

‘

1.1 Pattern Matching

In Computer Science, Pattern matching refers to the method of locating the occurrences of the pattern in a sequence. Output of pattern matching is the total number of occurrences of pattern P in a sequence S and all possible locations of a pattern P within a sequence S [1].

With time, new problems related to pattern matching are being defined. To deal with these problems in an efficient manner, several new data structures are being introduced and existing data structures are modified. Pattern matching algorithms can be distinguished from one another on the basis of the method used for searching the occurrence and the method used to achieve optimal time [2].

There are two techniques of Pattern Matching [3]:

- Single pattern Matching – A single pattern is searched for presence in a sequence.
- Multi pattern Matching – More than one patterns are searched simultaneously for presence in a sequence. It has high performance and usability than single pattern matching.

Pattern matching algorithms can be broadly classified into two main categories [3]:

- Exact pattern matching – It refers to finding the exact occurrence of the given pattern in the sequence.
- Approximate pattern matching – Approximate pattern matching allows for some errors or mismatches of some characters while finding the occurrence of the given pattern in a sequence [4]. Some of the main approaches used in approximate pattern matching algorithms are - Dynamic programming approach, Bit parallelism approach, Automata approach, Filtering and Automation Algorithms.

There are two steps involved in pattern matching algorithms [3]:

- Preprocessing phase – In this phase, the information is being collected for the purpose of optimization.
- Processing Phase – The information collected in preprocessing phase is being used to find the occurrences of the pattern in a sequence.

Pattern matching algorithms can be online or offline. In online algorithms, only pattern can be preprocessed while sequence cannot be preprocessed whereas in offline algorithms, both sequence and pattern can be preprocessed.

Some algorithms start searching for the pattern from the left side of the sequence and some from the right side of the sequence. Those which start from the left side are known as left-optimized algorithms and the ones that start from the right side are known as right optimized algorithms [3].

Each pattern matching algorithm has its own merits and demerits depending on the length of the pattern, length of the sequence and the technique used in that particular algorithm.

Application of Pattern Matching Algorithms includes [5]:

- Parsers
- Text processing
- Speech reorganization
- Spam filters
- Linguistic translation
- Digital libraries
- Screen scrapers
- Data compression
- Word processors
- Network intrusion detection
- Web search engines
- Information retrieval
- Natural language processing
- Computational molecular biology

- Feature detection in digitized images
- Computer virus detection

1.2 DNA

Deoxyribonucleic acid (DNA) is the storehouse of all information and genetic instructions used in the development and functioning of a cell. Thus, DNA sequences hold the code of life for every living organism. This information is normally encoded by the specific sequence of nucleotide bases i.e. adenine, guanine, cytosine and thymine (A, G, C, T) [6]. It is the linear order in which these bases are arranged that determines the properties of the cell.

DNA has “double helix” structure made up from two long interwoven strands. Each strand is made up of molecules known as nucleotides [7].

A nucleotide is made up of:

- A phosphate group
- A sugar called deoxyribose
- A base, which is one of the following:
  - A - Adenine,
  - T - Thymine,
  - C - Cytosine,
  - G - Guanine

The first two parts are identical in all nucleotides and form the backbone of the DNA strand.

There are five carbon atoms in the sugar molecule. These five carbon atoms are represented as C1’, C2’, C3’, C4’, C5’. With 1’ carbon, base is attached and with 3’ and 5’ carbons phosphate groups are attached. Sugar molecule is asymmetric and due to this asymmetry, it imposes an orientation on the backbone. The two ends that results due to this orientation are known as 3’ end and 5’end respectively [7].

DNA is double stranded because of base pair complementarily [7]. The two complementary base pairs are:

- A and T
- C and G

If one member of a pair is on one strand of DNA and the second member is on another strand, and both of them are aligned with each other, then the two can hybridize via hydrogen bonds. Hydrogen bonds are nothing but a weak attractive force between hydrogen and nitrogen or between hydrogen and oxygen. There are two hydrogen bonds between A and T as compared to C and G which is having three bonds between them, thus making C-G bonds stronger than A-T bonds [7]. The structure of the DNA is shown in Figure 1.1 [8].

The unit that is used to measure the length of the DNA is base pairs (bp).

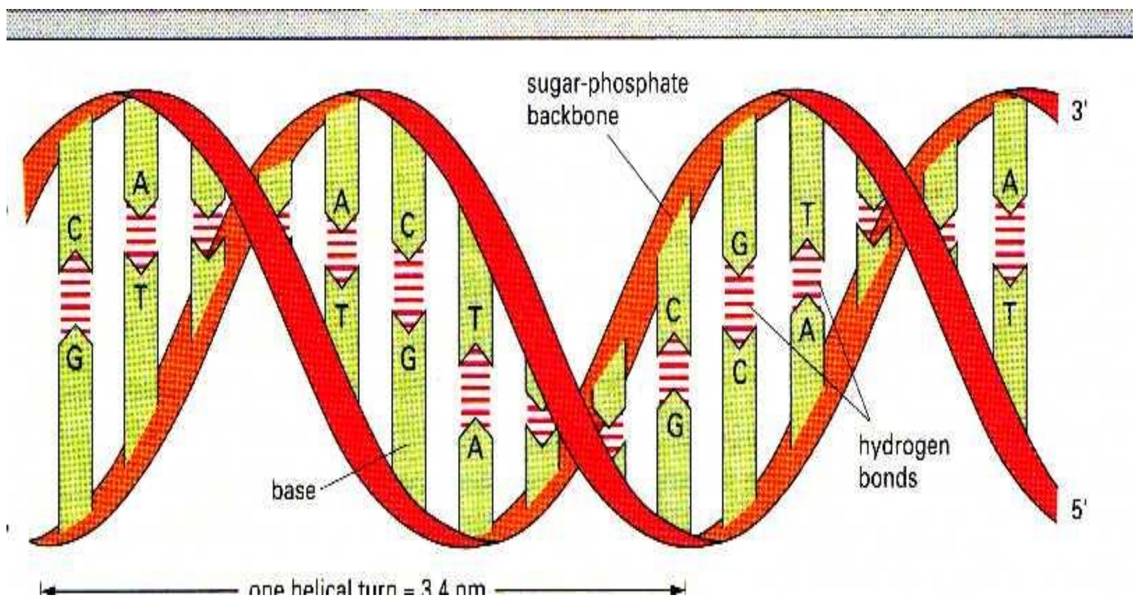

Figure 1.1: Structure of DNA

Sequences are presented by reading from left to right in 5’ to 3’ direction [9].

Every gene has coding region and non-coding region. Coding- region is known as exon and non- coding region is known as intron. In DNA, promoter region determines the initiation of the transcription process of a gene [8].

1.3 Pattern Matching in DNA Sequences

The amount of DNA that is being extracted from the organism is increasing at a faster rate. Because of the considerable increase in the amount of biosequence data, there is need to develop new methods to extract knowledge from the data. Pattern matching is a basic operation in finding knowledge from large amount of biosequence data. Pattern matching is used in computational biology to analyze the data related to protein and gene [10]. A particular pattern is searched for in a given DNA sequence. In any biological research, the most important step is searching for patterns in database. One example of such a database is GENBANK. The human DNA contains around 3Gbp [11]. The amount of DNA data being collected from an organism is increasing day by day in a non linear manner. With this increase in data, it is becoming difficult to obtain essential information from the DNA sequences. Thus efficient and fast pattern matching techniques are needed [3].

1.4 Significance of DNA Sequence Analysis

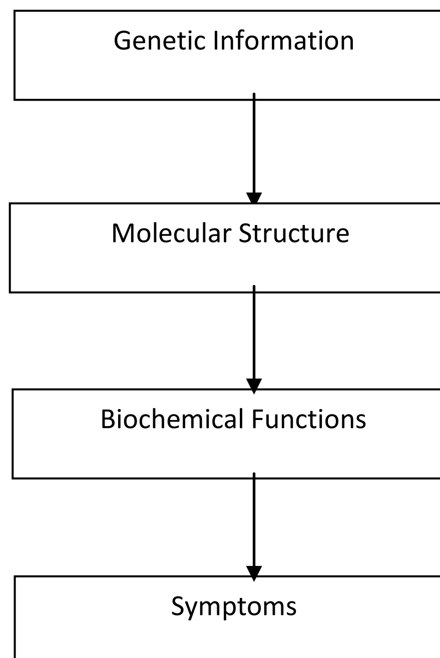

Figure 1.2: Central Paradigm of Bioinformatics

Genetic information is represented as one-dimensional but functionality of genes depend on the three dimensional structure. Structure determines biochemical characteristics of cell. By knowing the biochemical functions, symptoms of each cell can be easily known. This is shown in a flowchart in Figure 1.2 [12].

DNA is an important part of living things and thus knowledge of DNA sequences plays an important role in biological research. In medicine, knowledge of DNA can be used to diagnose the genetic diseases and then suggest treatment for the same [9].

1.5 Pattern Matching with Wildcard Characters

Wildcard refers to the special character that can be replaced by zero or more characters in a string. Wildcards are mostly used in regular expressions, SQL queries, Dictionary navigation etc. [13]. Pattern matching with wildcard gaps plays a significant role in biological sequence analysis in computational biology. Apart from bioinformatics [14], other applications of pattern matching with wildcard include information retrieval [15], dictionary query [16] etc.

The two examples in bioinformatics where pattern matching with wildcard gaps plays an important role are as follows:

1. It is known that in a DNA sequence, common promoter TATA box appears after the CAAT box at a gap of 30-50 characters [17, 18]. Hence pattern matching with wildcard gaps plays a vital role in analyzing such biological sequences.
2. To the specific locations of DNA, a protein is bind that regulates the transcription of the DNA into RNA. This protein is known as transcription factor. There are many transcription factors and can be classified into different families. The families are categorized on the basis of strings containing wildcards [19]. For instance, a transcription factor named Zinc Finger is having the signature:

   CYS¢¢CYS¢¢¢¢¢¢¢¢¢¢¢¢¢HIS¢¢HIS

   where CYS and HIS are the amino acids cysteine and histidine respectively.

1.6 Constraints

Various constraints that can be related to the pattern matching with wildcard gaps are as follows:

- Fixed wildcard gap - Fixed wildcard gap mean that the number of wildcard characters that can occur in pattern are fixed. While matching with a string, these wildcard characters can be replaced by any character from the alphabet under consideration [20].
- Variable wildcard gap – Variable wildcard gap means that the number of wildcard characters between two consecutive characters can be a range rather than a fixed number [20].
- Local length constraints - It is the constraint in the form of the range of length of wildcard characters between each two consecutive letters of the pattern. This gives flexibility to control queries [20].
- Global length constraints - Global length constraint is the constraint on the overall length of each matching substring of sequence with the given pattern [20].
- One-off condition- One-off condition means every positional index of a character in a sequence can be used at most once while matching with the given pattern [20]. Applying One-off condition makes the solution to satisfy Apriori property and also removes useless information.

More constraints lead to difficulty in achieving optimal solutions.

An example depicting variations of pattern matching is shown in Figure 1.3.
Figure 1.3a shows the case when there are no wildcard characters in the pattern. In this case jump from one character to the next character is consecutive while searching for the match.
Figure 1.3b shows the case when there is fixed wildcard gap in the pattern i.e. number of wildcard characters are fixed. In this case there is constant jump from one character to the next character while searching for the match.
Figure 1.3c shows the case when there is variable wildcard gap in the pattern i.e. number of wildcard characters between two consecutive characters of the pattern is a

range. In this case there is flexible jump from one character to the next character depending on the range while searching for the match.

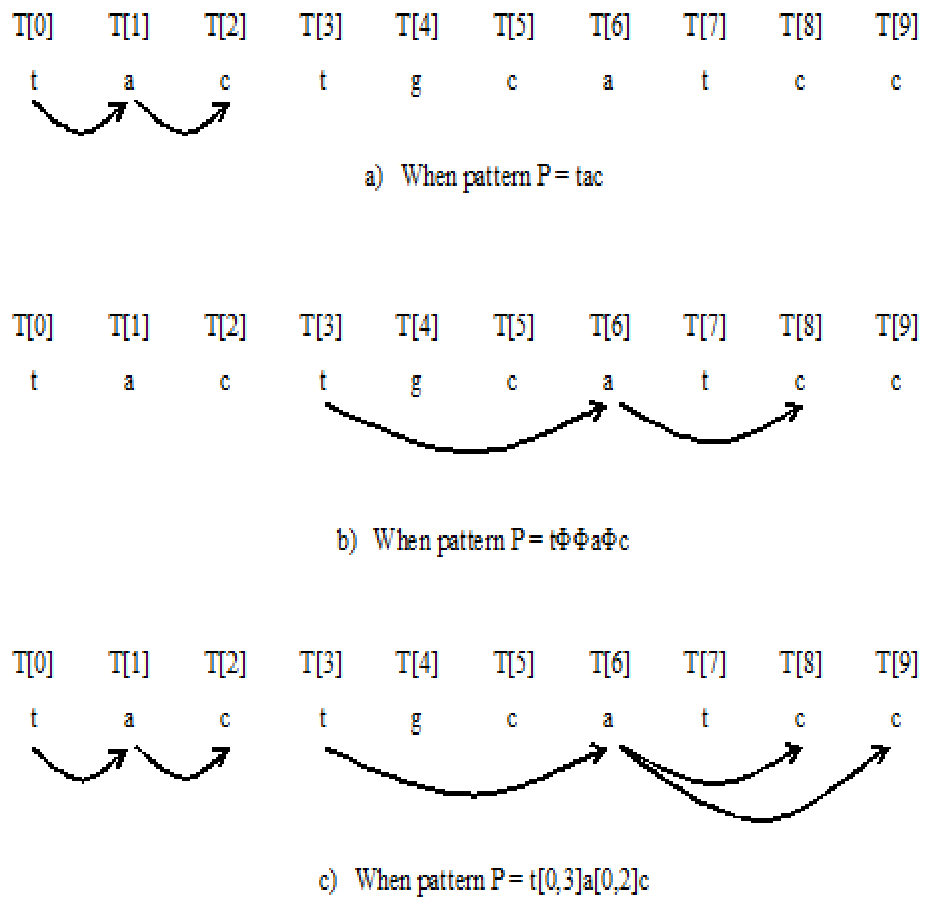

Figure 1.3: Example showing different cases of pattern matching

CHAPTER – 2
LITERATURE SURVEY

There exist a numerous algorithms for pattern matching problems. All of these algorithms differ in the way they search for the pattern, the data structure used by them and time taken by them to give the result.

2.1 Basic Concepts and Data Structure Used

2.1.1 Bit Parallelism

The concept of bit parallelism utilizes the ability of parallelism of bit operations in a computer word. Multiple values can be accommodated in a single computer word and all can be updated in just a single step. By doing that, the number of operations performed by an algorithm can be substantially reduced by factor of n where n being the number of bits in a computer word [21].

2.1.2 Trie

A tree consists of nodes connected to each other via unidirectional links [21]. Node from where the link starts is known as parent node and the node where it ends is known as the child node. Node not having any parent is known as the root node and nodes not having any child are known as the leaf node. If we attach labels to all the links present in the tree, the tree is known as labeled rooted tree. These labels are from a particular alphabet ∑, definition of which varies depending on the problem/application.

If we associate this labeled rooted tree to a set of strings, it is known as trie. Structure of trie is shown in Figure 2.1.

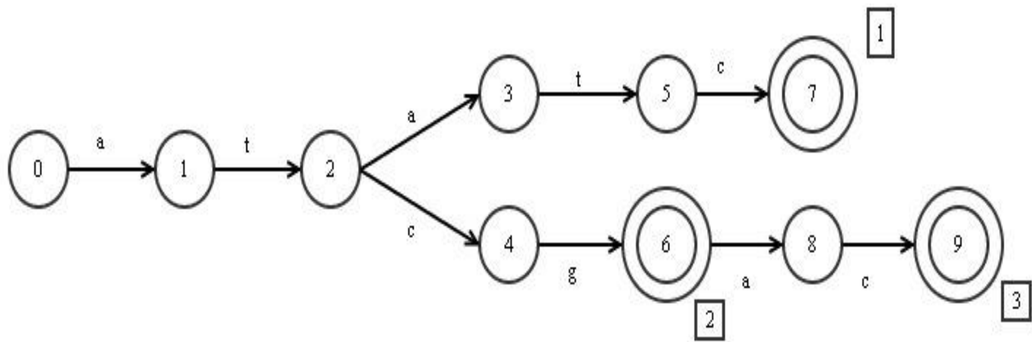

Figure 2.1 Trie for the set of strings, S= {atatc, atcg, atcgac}

2.1.3 Automata

Finite automata can be defined as the machine that captures all the possible states and transitions while processing the input symbols.

Depending on the fact whether a machine can have only one state at a particular time or can exist in multiple states at the same time, it can be categorized into deterministic finite automata and non-deterministic automata respectively [22].

If all the transitions are properly labeled from the set of alphabet ∑, automata can recognize strings that label path from the initial state to the final state of automata [21].

Example of DFA and NFA has been shown in Figure 2.2 and Figure 2.3 respectively to clearly distinguish them.

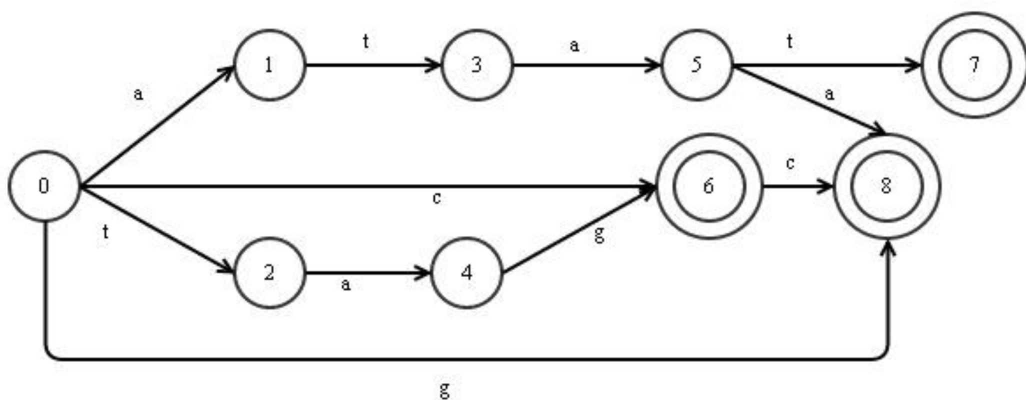

Figure 2.2: Deterministic Automata

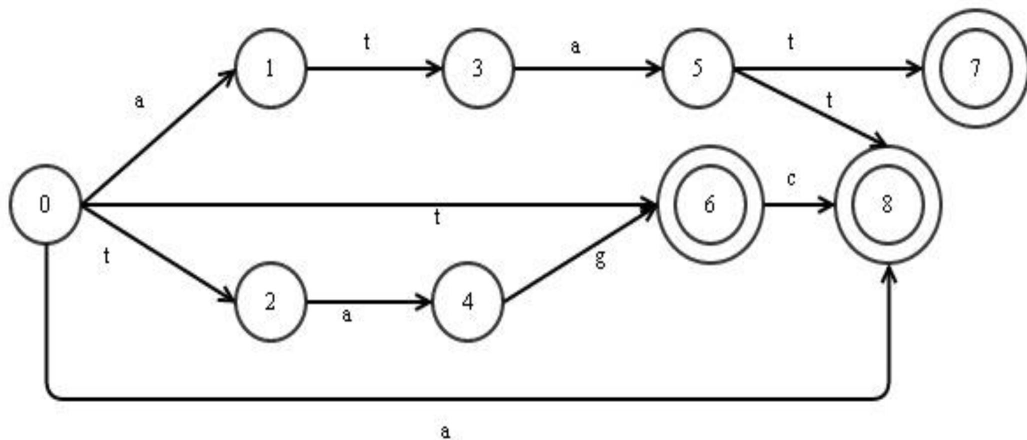

Figure 2.3 Non-Deterministic Automata

In both the figures, 0 is the start state and states with double circle are final states. Clearly Figure 2.2 is deterministic automata as for a particular character no state is leading to multiple states. Similarly Figure 2.3 is non-deterministic automata as character ‘t’ from state 0 is leading to two states – 2 and 6.

Further automata can be cyclic or acyclic in nature.

2.1.4 Suffix Tree

It is similar to trie data structure. It stores all the suffixes of a given string [23]. It is a rooted directed tree with the following properties [24]:

- Number of leaves is equal to the length of the given string.
- Each edge is labeled. Label must be a substring of the given string.
- All nodes except root node and leaves should have at least two children.
- Edges coming out from the same node must not have label beginning with the same character.
- If we traverse the suffix tree from the root node to any leaf node, concatenation of the labels of the edges in the path of traversal represents the suffix of the given string.

Suffix tree is built by considering all the possible suffixes of the string as individual words and building a compressed trie for these words.

Example of suffix tree for the string "banana" is shown in Figure 2.4. All the suffixes for the string "banana" are – 'banana', 'anana', 'nana', 'ana', 'na', 'a' and ''.

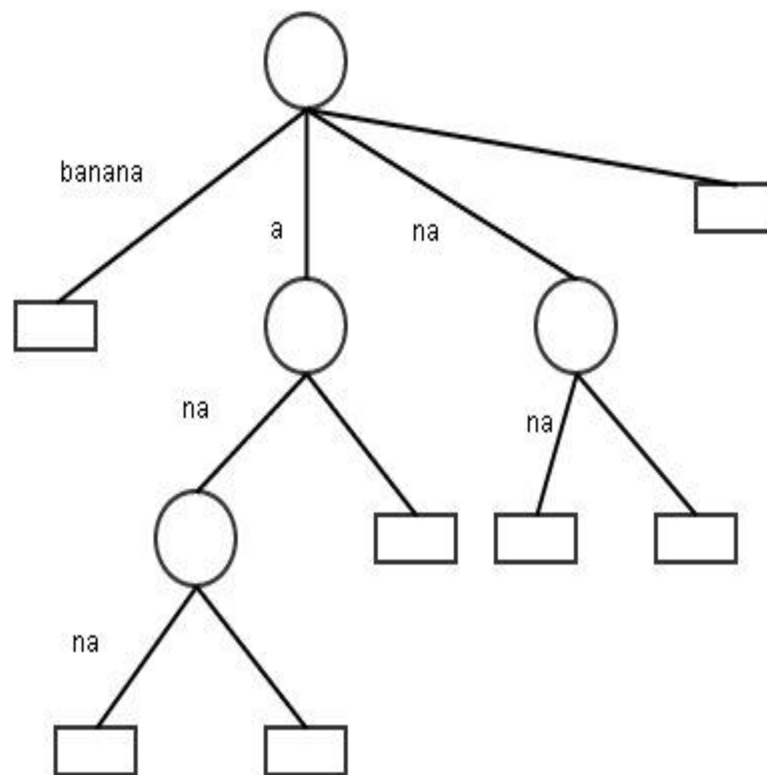

Figure 2.4: Suffix tree for the string "banana"

2.2 Different Matching Strategies

There are mainly three different ways of searching for pattern in a text. Pattern is searched for in a given string using a sliding window. Size of the window is equivalent to the size of the pattern. Direction of movement of a window in a string is from left to right. Pattern is matched inside a window. Different Approaches along with the algorithms that use that particular approach are shown in Figure 2.5 [21].

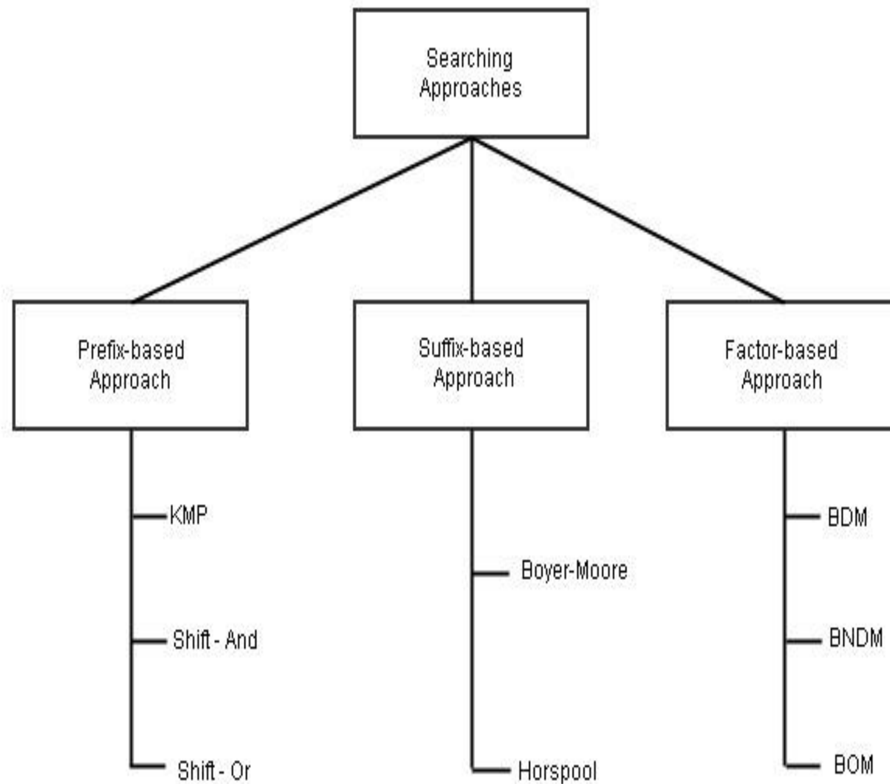

Figure 2.5: Categorization of Searching Approaches

2.2.1 Prefix-based Approach

In prefix-based approach forward search is done to find the longest suffix of the window that is also the prefix of the pattern [21]. There are three algorithms that fall under this category: KMP algorithm, Shift-And algorithm and Shift-or Algorithm.

KMP algorithm [25] – It uses deterministic finite automata. It updates the longest prefix of the pattern that matches the suffix of the string in window after each character read.

Shift-and algorithm [4] and Shift-or algorithm [26] – These algorithms use non-deterministic finite automata and works on the bit – parallel technique. All the possible prefixes of the pattern that can match all the possible suffixes of the string in window are being maintained in a set and updated after each character read.

2.2.2 Suffix-based Approach

In suffix-based approach, backward search is done to find the longest suffix of the window that is also the suffix of the pattern [21]. By doing this, some characters can be avoided from being read, thus improving the performance. The algorithms that use this approach are Boyer-Moore algorithm and Horspool algorithm.

Boyer-Moore (BM) algorithm [27] – It involves computing the three functions. These three functions are used for shifting i.e. to determine the safe jumping distance.

Horspool algorithm [28] – The complexity of BM algorithm was in computing the three functions. Horspool changed the third function such that it is more efficient to compute it.

2.2.3 Factor-based Approach

In Factor-based approach, backward search is done to find the longest suffix of the window that is also a factor of the pattern [21]. The complexity of this approach lies in identifying the factors of the pattern. The algorithms that use this approach are Backward Dawg matching algorithm, Backward Non-deterministic Dawg matching algorithm and Backward oracle matching algorithm.

Backward Dawg Matching (BDM) algorithm [29] – In order to search the factors of the pattern, it uses suffix automation.

Backward Non-deterministic Dawg Matching (BNDM) algorithm [30] - In order to search the factors of the pattern, it uses bit parallelism. BNDM is memory efficient than BDM. It is basically simulation of non deterministic automation representing suffixes of reverse pattern.

Backward Oracle Matching (BOM) algorithm [31] – It is a slight modification of factor based approach. The complexity of the algorithm BDM lies in building the suffix automation. The algorithm BOM uses the automation named factor oracle which is much simpler as compared to suffix automation.

2.3 Variations in Traditional Pattern Matching Problem

2.3.1 Fixed Wildcard Characters

The concept of pattern matching with wildcard was first introduced in [32] by Fisher and Paterson in which the location of wildcard characters in a pattern is fixed. The algorithm given by Fisher and Paterson was deterministic in nature.

Muthukrishnan and Palem were able to slightly improve the algorithm given by Fisher and Paterson by reducing the constant factor [33].

Time efficiency of the matching result of the algorithm by Fisher and Paterson was improved by Indyk who gave randomized algorithm involving convolutions [34]. A new randomized technique was given by Indyk to calculate the Boolean products. Time complexity of this algorithm is O (n log n) where n is the length of the string.

Kalai [35] slightly improved the time efficiency of the matching result of the algorithm given by Indyk. Algorithm given by Kalai is also a randomized algorithm and involves a single convolution. Time complexity of this algorithm is O (n log m) where n is the length of the string and m is the length of the pattern.

Cole et al. [17] put a restriction on the overall number of wildcard that can occur in a pattern while allowing for any number of wildcard characters in between the two consecutive characters of the pattern.

2.3.2 Variable Wildcard Gap

In [6, 17], user was able to specify the range of wildcard gap between consecutive characters of the pattern, but that range was fixed for all the consecutive characters of pattern. E.g. A(0,3)T(0,3)A(0,3)C has fixed gap range of (0,3). In [17], Manber and Baeza-Yates made use of suffix array data structure to solve the problem, thus reducing it to the two-dimensional orthogonal range queries problem.

Navarro and Raffinot [36] eliminated the restriction of fixed user specified range, thus allowing variable user specified gap range. E.g. A(0,1)T(0,3)A was allowed in [36]. Navarro and Raffinot proposed two algorithms. The algorithms are not based on

regular expression technique, thus making them a bit faster. The first algorithm reads each character of string exactly once. The second algorithm can skip some characters of string from being read thus making it efficient, but in some cases, it can end up reading the characters of strings more than once. So depending on the particular case, one of the two algorithms is run.

Min et al. [37] deals with the same problem definition as in [36] with global length constraint added. They proposed the algorithm PAIG. There are three variations of this algorithm - PAIG(S), PAIG (RS) and PAIG (RST). PAIG(S) stands for PAIG simple. The data structure being used in PAIG is simple look-up tables. PAIG (RS) stands for PAIG reduced space. In PAIG (RS), memory sharing mechanism is used which reduces the space complexity. PAIG (RST) stands for PAIG reduced space and time. In PAIG (RST), an alternative data structure is being employed which reduces both time and space complexity.

2.3.3 One-off Condition

In addition to the global constraint, the concept of one-off condition was taken into consideration in some algorithms.

SAIL algorithm [20] – It consumes a lot of time for the large pattern length. Three main steps involved in this algorithm are:

1. Location: It searches for the position of the last alphabet of Pattern in Sequence by considering the global constraint.
2. Forward: This phase eliminates all those solutions that do not satisfy local constraints and gives the underlying matching positions.
3. Backward: This phase selects one optimal solution out of all possible solutions.

RSAIL algorithm [38] - Being a heuristic algorithm, SAIL has a problem of left-optimization as it chooses the left-most letters. It may lose occurrences by getting trapped in local optima. SAIL offers the completeness under a restriction that pattern should not have recurring characters.

To eliminate this problem with SAIL algorithm, RSAIL was proposed. The idea behind RSAIL is as follows:

1. If pattern is not having recurring tail characters, SAIL is called.
2. If pattern is having recurring tail characters, convert it into a pattern having no recurring tail characters and call SAIL.

Time complexity of RSAIL is same as that of SAIL.

BPBM algorithm [39] - It is based on bit-parallel technology. Two nondeterministic finite state automations (NFAs) are used. One is for identifying all of the pattern suffixes, and another one is used to fasten the scanning process by eliminating useless sequences.

PST algorithm [23] - Yingling LiU et al. proposed PST algorithm. PST (parallel suffix tree) algorithm is based on multiple suffix trees.
The algorithm steps are given as:
Step1: Sequence S is divided into K parts by the cutting process;
Step2: Suffix tree is constructed for each part;
Step3: Multiple suffix trees are processed in parallel in order to get the matching locations for the pattern.

PMW algorithm [40] - Jipeng Qiang et al. proposed an algorithm PMW. This algorithm relies on the reversed Aho-Corasick automation for matching the sub-patterns. Horspool algorithm is used to fasten the scanning process by eliminating useless sequences. For each sub-pattern, an optimal occurrence is chosen. This algorithm is left-optimized.

HSO algorithm [41] - Wu Y et al. proposed a new data structure Nettree [42]. When global length constraints are not considered, PAIG is inefficient because it recalculates some local constraints. So a new data structure namely, Nettree was introduced and a heuristic algorithm HSO based on this data structure was proposed.

WOW algorithm [43] – Guo et al. introduced a new data structure named WON-Net and proposed a heuristic algorithm WOW based on this data structure. Three strategies are mentioned- LMO (Left Most Optimum), RMO (Right Most Optimum)

and CMP (Centralization Measure Pruning). One of the three strategies is used depending on the calculation of some parameter.

2.3.4 Approximation Pattern Matching

SAIL- Approx Algorithm [44] - The algorithm SAIL was extended in [44] to allow for some errors i.e. approximate pattern matching. The concept of dynamic programming is being applied in this algorithm.

2.4 Comparison of Various Algorithms Involving One-off Condition

Various algorithms to solve the problem of maximal pattern matching with length constraints and one-off condition have been compared in Table 1.1 on the basis of data structure, time and space complexities.

Meaning of various symbols used in the comparison table is as follows:

n – Length of the sequence
m – Length of the pattern
f - Frequency of occurrence of pattern's last character in the sequence
W - Maximum gap between consecutive letters of the pattern
l – Maximum allowed length of the occurrence (Global Length Constraint)
c - Number of parts into which sequence is divided
num – total number of occurrences of the pattern in a sequence
α – Total number of occurrences of sub patterns in a sequence
A – Sum of lower limits of gap range
B - Sum of upper limits of gap range
s – Number of sub patterns in a pattern
L – Number of characters in the last sub pattern of the pattern
B/w – Number of machine words to store each bit mask

Table 2.1: Comparison of Algorithms

Algorithm	**Data Structure**	**Time Complexity**	**Space Complexity**
SAIL	Search Table	O(n+flmW)	O(lm)
RSAIL	Search Table	O(n + flmW)	O(lm)
PST	Suffix Tree	O(n+m+num+n/c)	O(2n/c)
BPBM	Non-deterministic Finite automata	O((Bm+n+f(l+s − 1))(B/w))	O((m+L+2s + 4) (⌊ B/w⌋))
PMW	Aho-Corasick Automation	O(m+n+ f(l+α))	O(m+A)
HSO	Nettree	$O(Wn(n+m^2))$	O(Wmn)
WOW	WON-Net	$O(Wmn+mn^2)$ by LMO/RMO $O(Wmn+mn^3)$ by CMP	O(mn)

CHAPTER – 3
PROBLEM STATEMENT

3.1 Problem Definition

Given a biological sequence S, a pattern P along with user defined local and global constraints, our goal is to find the maximum number of substrings of sequence S that matches the pattern P satisfying the local and global constraints under the one-off condition.

Definition 1: A biological sequence S is defined as

$$S= s_0 s_1 s_2 \ldots s_i \ldots s_{n-1}$$

where n is length of the sequence S and $s_i \in \{a, t, c, g\}$ $\forall$ i where $0 \leq i < n$.

Example 1: A sequence S = aaattcgatgggcat is a biological sequence with length, n = 15.

Definition 2: A pattern P is defined as

$$P = p_0 [l_0 , u_0]\ p_1[l_1 , u_1]\ p_2[l_2 , u_2] \ldots [l_{j-1} , u_{j-1}]\ p_j[l_j , u_j] \ldots [l_{m-2} , u_{m-2}]\ p_{m-1}$$

where m is length of the pattern P without wildcards and p_i ε {a, t, c, g} $\forall$ $0 \leq j < m$. Here $[l_j , u_j]$ is the range of wildcard gap allowed between the pattern characters p_j and p_{j+1}. l_j depicts the lower limit on the number of wildcard characters and u_j depicts the upper limit on the number of wildcard characters. This wildcard gap specified between every two consecutive characters of P is called local constraint.

Example 2: P = a[0,3]t is a pattern with length i.e. m = 2. Here between the characters ‘a’ and ‘ t’, 0 to 3 wildcard characters are allowed.

Definition 3: Global length constraint is defined as the constraint on the overall length of the substring of the sequence that matches the pattern. It is defined as [min, max]. ‘min’ and ‘max’ depicts the minimum and maximum allowable overall length of the substring of the sequence that matches the pattern respectively.

Definition 4: m and n being the length of the pattern and the sequence respectively, if there exists positional indices $o_0o_1o_2\ldots o_{m-1}$ in a sequence S= $s_0s_1s_2\ldots s_{n-1}$ such that characters against those positional indices matches the characters of the pattern P = p_0 $[l_0 , u_0]$ $p_1[l_1 , u_1]$ $p_2[l_2 , u_2]$ ...$[l_{m-2} , u_{m-2}]$ p_{m-1}, i.e.

$$s_{o_i} = p_i \ \forall\, i \text{ where } 0 \le i \le m-1$$

then $(o_0o_1o_2\ldots o_{m-1})$ is called an occurrence of a pattern in a sequence.

Example 3: Suppose sequence S = atataaa and pattern P = a[0,3]t[0,5]a. All the possible alignments of pattern P with sequence S satisfying the local constraints are given in Table 3.1.

Table 3.1: Different Alignments of Pattern with Sequence in Example 3

	0	1	2	3	4	5	6	
S	a	t	a	t	a	a	a	
P	a	t	a					(0,1,2)
P	a	t	-	-	a			(0,1,4)
P	a	t	-	-	-	a		(0,1,5)
P	a	t	-	-	-	-	a	(0,1,6)
P	a	-	-	t	a			(0,3,4)
P	a	-	-	t	-	a		(0,3,5)
P	a	-	-	t	-	-	a	(0,3,6)
P			a	t	a			(2,3,4)
P			a	t	-	a		(2,3,5)
P			a	t	-	-	a	(2,3,6)

So there are total of 10 possible occurrences of pattern P in sequence S : {(0,1,2), (0,1,4), (0,1,5), (0,1,6), (0,3,4), (0,3,5), (0,3,6), (2,3,4), (2,3,5), (2,3,6)}

Example 4: Suppose in example 3, global length constraint [3, 5] is given. In this case, we are left with only six possible occurrences- {(0,1,2), (0,1,4), (0,3,4), (2,3,4), (2,3,5),(2,3,6)} as the occurrences (0,1,5), (0,3,5) are having length 6 and the occurrences (0,1,6), (0,3,6) are having length 7 whereas the maximum possible length allowed is 5.

Definition 5: If every positional index of a character in a sequence can be used at most once while matching with a pattern, then such a set of occurrences is said to follow the one-off condition. The solution {occ_1, occ_2...occ_i} is said to follow the one- off condition if and only if

$$occ_1 \cap occ_2 \ldots \cap occ_i = \phi$$

where i is the total number of occurrences in the solution.

Example 5: After applying the one-off condition in example 4, the possible solutions are: {(0,1,2)}, {(0,3,4)}, {(2,3,4)}, {(0,1,4), (2,3,5)}, {(0,1,4), (2,3,6)} . Since our problem is to find the maximum number of possible occurrences, we should get as a solution either {(0,1,4), (2,3,5)} or {(0,1,4), (2,3,6)} as both solutions contain 2 occurrences whereas rest of the possible solutions contain only single occurrence.

3.2 Gap Analysis

For the problem stated in above section, there is no complete solution developed so far. All existing solutions dealing with this problem are based on greedy matching strategies. Since this problem is computationally infeasible, it is difficult to develop complete matching strategies. The problem of maximal pattern matching with flexible wildcard gaps and length constraints under the one-off condition belongs to the category of optimization problem. So focus is to improve the matching efficiency as well as the quality of solutions.

3.3 Proposed Objective

The main objectives to address the above stated problem are as follows:

- To study the various data structures used in pattern matching.
- To analyze and compare different techniques used in existing algorithms designed for the above stated problem.
- To propose a new algorithm for the problem of maximal pattern matching with flexible wildcard gaps and length constraints under the one-off condition.
- To validate the new algorithm on biological data.

3.4 Methodology Used

To achieve the objectives stated in the above section, following methodology has been used:

- Compare the existing algorithms for the problem on the basis of data structure, approach, and time and space complexities.
- Choose a particular data structure and propose a new technique based on the data structure chosen.
- Implement the new algorithm for maximal pattern matching with flexible wildcard gaps and length constraints under the one-off condition.
- Validate the algorithm on real world biological data
- Compare the results of new algorithm with the results obtained from the existing algorithms on the same dataset.

CHAPTER – 4
ALGORITHM

4.1 Data Structure

Proposed algorithm is based on the Nettree data structure [42]. Nettree data structure is non-linear. Nettree is graph cum tree with one or more roots. Also it is acyclic in nature. Nodes can have zero or more parents except those at the root level. Similarly nodes can have zero or more children except those at the leaf level.

Structure of Nettree node is shown in Figure 4.1.

data
next
degree_parents
degree_children
parents
children
num_root_paths

Figure 4.1: Structure of Nettree Node

There are seven fields in the structure of **Nettree nodes:**

- data contains the position of the character in the sequence.
- next is the pointer that points to the immediate next node of the node.
- degree_parents represents the number of parents of the node.
- degree_children represents the number of children of the node.
- parents is a pointer array that contain pointers to all the parents of the node. This array is of size ‘degree_parents’.
- children is a pointer array that contain pointers to all the children of the node. This array is of size ‘degree_children’.

- num_root_paths represent the total number of paths from this node to the root level nodes.

Structure of Nettree level is shown in Figure 4.2. Total number of levels in Nettree is equal to total number of subpatterns in a given pattern. Here subpattern refers to the subpatterns that are separated by wildcard gaps in a pattern. For instance, if pattern is a[0,3]t[2,3]c, then there are three subpatterns of this pattern i.e. ‘a’, ‘t’ and ‘c’. There can be any number of nodes at each level of Nettree depending on the sequence and pattern. The very first level is known as root level and the last level is known as leaf level. The concept of roots and leafs is similar to that of a tree.

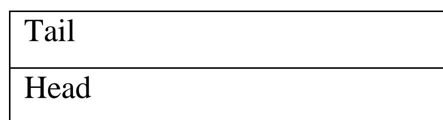

Figure 4.2: Structure of Nettree Level

There are two members in **Nettree Level:**

- Head- First node of the level is pointed by this pointer.
- Tail – Last node of the level is pointed by this pointer.

From one particular node at leaf level to a particular node at root level, there can be multiple paths.

Num_root_paths field of a Nettree node contains number of all possible paths from that node to the nodes at root level. In order to calculate num_root_paths, we start from the root level. For all the nodes at root level, value of num_root_paths is equal to one. For all other levels, value of num_root_paths is equal to the sum of num_root_paths of all the parents of that node.

If we will sum the num_root_paths of all the nodes at leaf level, we will get the total number of possible paths we can get by traversing from leaf level to root level in Nettree.

Total number of possible paths depicts the total number of occurrences of the pattern in a sequence.

While creating Nettree, local constraints are taken care of i.e. the occurrences that we get after traversing Nettree satisfies the local constraints. However, occurrences outputted by traversing Nettree do not satisfy global constraints and the one-off condition. It just provides the solution for the problem of pattern matching with independent wildcard gaps.

If we traverse the Nettree from root to leaf level, we get position of all occurrences of the pattern in a sequence.

Nettree is being created according to the sequence S and Pattern P. Sequence S is scanned from left to right. Nodes and relation between nodes will be created according to the following rules:

Rule 1. Creation of nodes of root level

If $s_i = p_0$, create and add node to the tail of the level one. In case it is the first node of the level, make head point to this node.

Rule 2. Creation of nodes other than root level

If $s_i = p_j$ where j != 0 and the distance between i^{th} and the jth level nodes is in accordance to local constraints, create and add node to the tail of the $j + 1^{th}$ level.

Rule 3. Creating relation between nodes of different levels

If the distance between the node created at a level and the nodes at one level up satisfies the local constraints, create a parent-child relation between nodes.

Example:

Let sequence be atataaa and pattern be a[0,3]t[0,5]a

Step by step construction of Nettree for this problem is shown in Figure 4.3. Final Nettree created is shown in Figure 4.4.

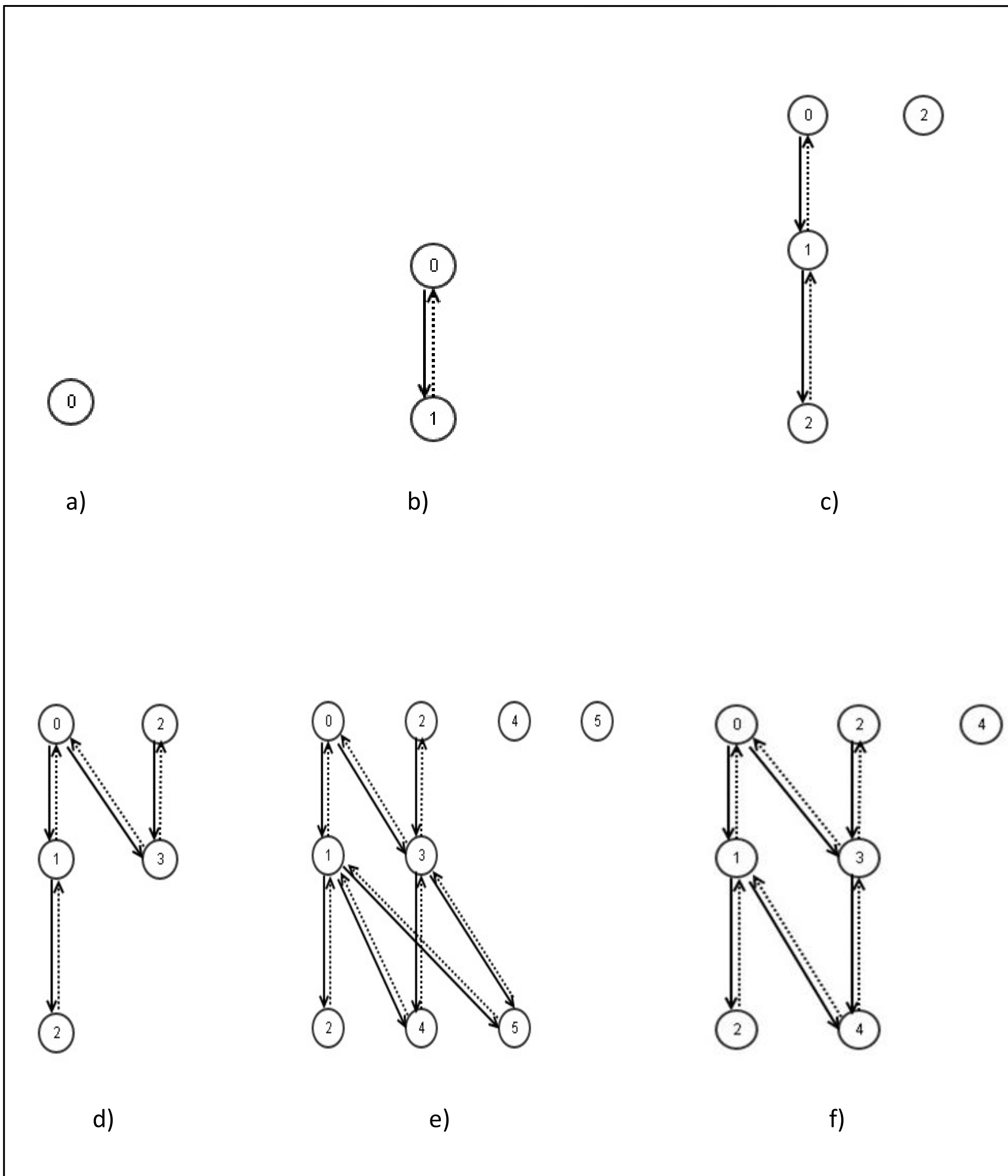

Figure 4.3: Step-by-Step Creation of Nettree

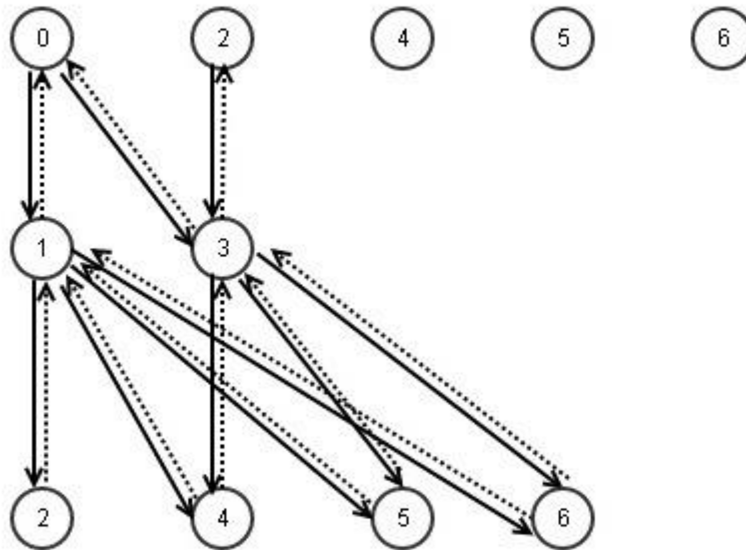

Figure 4.4: Final Nettree

On traversing this Nettree, from the leaf level up to the root level, we get all the possible paths.

In this example, total number of paths=10

And those paths are:

0 1 2 2 3 4 0 3 4 0 1 4 2 3 5 0 3 5 0 1 5 2 3 6 0 3 6 0 1 6

Complexity

Time and space complexity of creating Nettree data structure according to the given sequence and pattern is O(W*m*n) and O(W*m*n) where m is the length of the pattern, n is the length of the sequence and W is the maximum gap between consecutive letters of the pattern. Maximum depth of Nettree is m. Also there cannot be more than n nodes at one level as length of sequence is n. Since maximal gap is W, so for each node, there can be maximum of W parents possible i.e. next W positions in a sequence. Hence time and space complexity of Nettree is O(W*m*n) and O(W*m*n) respectively.

4.2 Algorithm Description

Definition 6: number_levels[node.data] contains number of different levels at which the positional index of node i.e. node.data is occurring.

Definition 7: Considering the paths from a particular node 'nod' to the root level, the sum of the value of the number_levels of the nodes in the path that contains less number of those nodes that occur at various different levels as compared to other possible paths is called min_occurence of the node 'nod'.

Property 1: For a node except those at the root level, the minimum value of the min_occurence amongst all the parents of the node plus the number_levels value of the node itself is known as min_occurrence of the node. For a node at root level, min_occurence is the value of the number_levels of that particular node.

node.min_occurence = min(node.parents[i].min_occurrence) + number_levels[node.data]

where $1 \leq i \leq$ node.num_parents

Definition 8: Considering the paths from a particular node 'nod' to the root level, the sum of the value of the number_levels of the nodes in the path that contains more number of those nodes that occur at various different levels as compared to other possible paths is called max_occurence of the node 'nod'.

Property 2: For a node except those at the root level, the maximum value of the max_occurence amongst all the parents of the node plus the number_levels value of the node itself is known as max_occurrence of the node. For a node at root level, max_occurence is the value of the number_levels of that particular node.

node.max_occurence = max(node.parents[i].max_occurrence) + number_levels[node.data]

where $1 \leq i \leq$ node.num_parents

Algorithm 1: MOGO

Input: Sequence S, pattern P and global constraint [min, max]
Output: Set of occurrences

Method:

```
1:  Build the Nettree  for the sequence S and the pattern P
2:  Remove nodes having no possible path to any leaf from the Nettree
3:  for l ϵ number of leaves at m^th level down to 1 step -1
4:          for nod ϵ all nodes of the Nettree
5:                  Number_levels[nod.data]+=1
6:          end for
7:          for nod ϵ all nodes of the Nettree
8:                  calculate nod.min_occurrence and nod.max_occurrence
    according to property 1 and property 2
9:          end for
10:         for nod ϵ all nodes of the Nettree
11:                 calculate the possible roots for nod satisfying global constraints
12:         end for
13:         if l satisfies global constraints
14:                 occ = OOCL(l, Nettree)
15:                 solution=solution U occ
16:                 Nettree = Nettree – occ
17:         end if
18: end for
19: return solution
```

Algorithm 2: OOCL

Input: Nettree, Leaf l

Output: An occurrence containing Leaf l at last position

Method:

```
1:  best_parent = l.parent[l.num_parents]
2:  for r ϵ l.num_parents down to 1 step -1
3:          if l.parent[r] satisfies global constraints
4:                  if l.parent[r].min_occurrence < best_parent.min_occurrence
5:                          best_parent = l.parent[r]
6:                  elif l.parent[r].min_occurrence == best_parent.min_occurrence
7:                          if l.parent[r].max_occurrence <=
    best_parent.max_occurrence
```

```
8:                              best_parent = l.parent[r]
9:                      end if
10:             end if
11:     end if
12: end for
13: if best_parent.degree_parents != 0
14:     OOCL(Nettree, best_parent)
15: end if
16: return occurrence
```

MOGO (Maximum Occurrences with Global length constraints and One-off condition)

In line 1, Nettree is being created according to the sequence S and Pattern P. In line 2, nodes that cannot be a part of any path from leaf to the root are being removed. For the purpose of removing such nodes, each node of the constructed Nettree is inspected. From lines 3 to 18, for each node - number_levels, possible roots satisfying global constraint, min_occurrence and max_occurrence are being calculated and OOCL is called iteratively for each leaf satisfying the global constraint in order to get an optimal occurrence containing that leaf. In lines 4 to 6, nettree_levels is calculated. In lines 7 to 9, min_occurrences and max_occurrences for each node is calculated. In lines 10 to 12, for all nodes, all the possible root nodes satisfying the global constraints that can be reached from node in consideration are being calculated. In line 13, node is checked whether it is satisfying global constraints. If it satisfies global constraints, in line 14 the algorithm OOCL is called for that node.

OOCL (Optimal Occurrences Containing Leaf)

This algorithm works in a recursive manner by finding the best parent of the node for which it is called till it reaches the root level. It chooses the parent having minimum value of min_occurrence as the best parent. In case of clash of the minimum value of min_occurrence, it chooses the one having the minimum value of max_occurrence. This algorithm returns a single occurrence containing the node with which it was called by MOGO at last position.

4.3 Complexity Analysis

The space complexity of storing the Nettree is O (W*m*n). Hence the space complexity of MOGO is also O(W*m*n) where m is the length of the pattern, n is the length of the sequence and W is the maximum gap between consecutive letters of the pattern.

The time complexity of lines 1 and 2 of MOGO is O(W*m*n). Lines 4 to12 of MOGO are having time complexity of O (W*m*n).Time complexity of OOCL is O(W*m). Complexity of lines 15 and 16 of MOGO is O (m). Thus complexity of line 3 through 18 is O((W*m*n)*n/m) i.e. O(W*n*n).Thus overall time complexity of MOGO is O(W*m*n + W*n*n) i.e. O(W*n*(n+m)).

4.4 An Illustration Example

For the sequence S = aatattaat and the pattern P = a[0, 2]t[0, 1]a[0, 3]t and the global length constraint of [4, 10] , the nettree being created is shown in Figure 4.5.

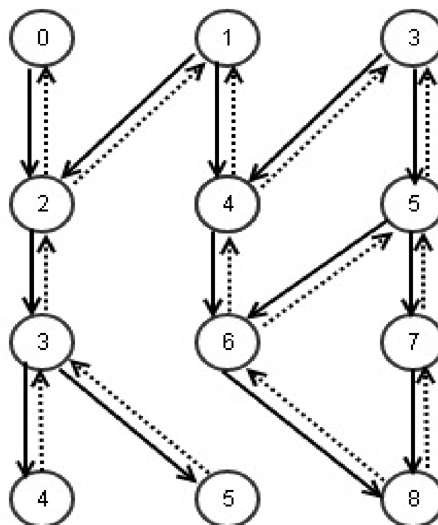

Figure 4.5: A Nettree

In Figure 4.5, solid line and dotted line depicts the parent-child and child-parent relation respectively. Number_levels calculated from this Nettree is shown in Table 4.1.

Table 4.1: An Instance of Number_levels Array

Array Index	0	1	2	3	4	5	6	7	8
Element	1	1	1	2	2	2	1	1	1

min_occ and max_occ calculated for each node is shown in Figure 4.6.

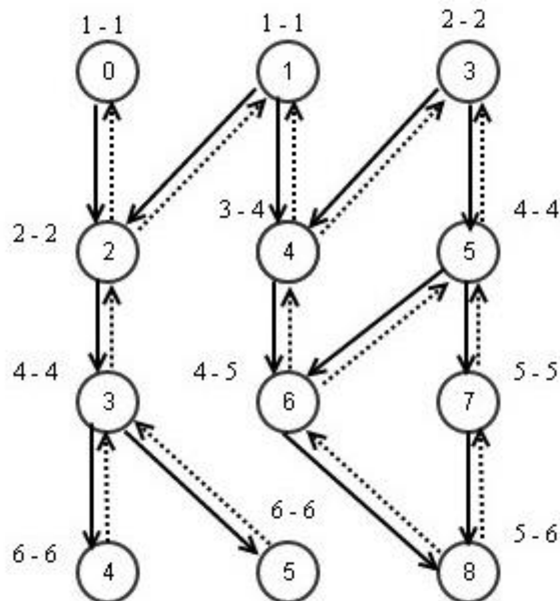

Figure 4.6: Nettree with min_occ and max_occ of each node

Here the value before '–' represents value of min_occurrence and the one after '–' represents the value of max_occurrence. For e.g. if its 3- 4, this means value of min_occurrence is 3 and that of max_occurrence is 4. Amongst all the parents of 8, 6 is having the minimum min_occurrence value. So 6 is the best parent. Now for 6, 4 is the best parent. Similarly for 4, 1 is the best parent. In this way we get an occurrence (1, 4, 6, 8). These nodes are then removed from the Nettree. Figure 4.7 shows removal of nodes and the Nettree recreated after removal of nodes is shown in Figure 4.8.

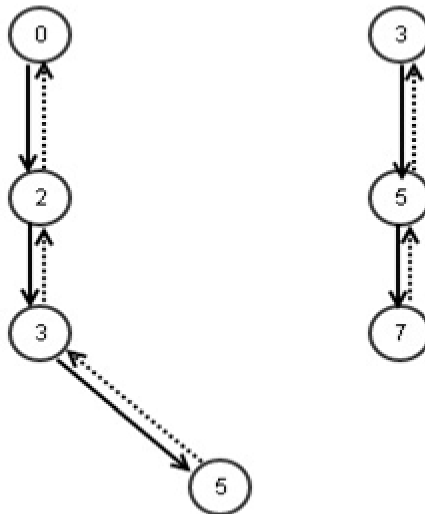

Figure 4.7: Removal of nodes from Nettree

Same operation is then performed on the next leaf node.

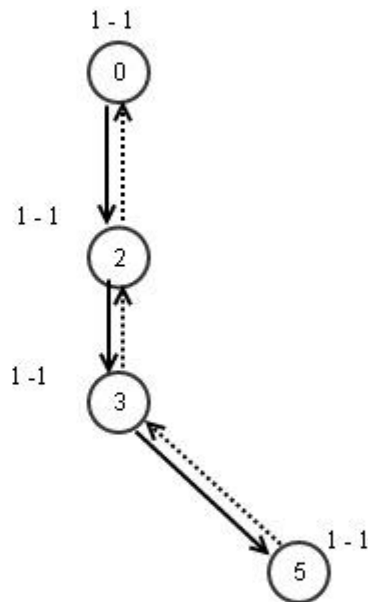

Figure 4.8: Updated Nettree

In this case we get (0, 2, 3, 5) as another occurrence. Then these nodes are again removed from the Nettree and we are left with no other leaf nodes. We get {(1, 4, 6, 8), (0, 2, 3, 5)} as the final solution.

CHAPTER – 5

IMPLEMENTATION AND EXPERIMENTAL RESULTS

5.1 Configuration and Architecture

Experiment is performed on machine with configuration Intel(R) Core(TM)2 Duo CPU T6500@2:10 GHz, 3 GB of RAM and Windows 7 OS. Algorithm has been implemented in Python 3.2. For interface CGI (Common Gateway Scripting) scripting has been used. Architecture is shown in Figure 5.1.

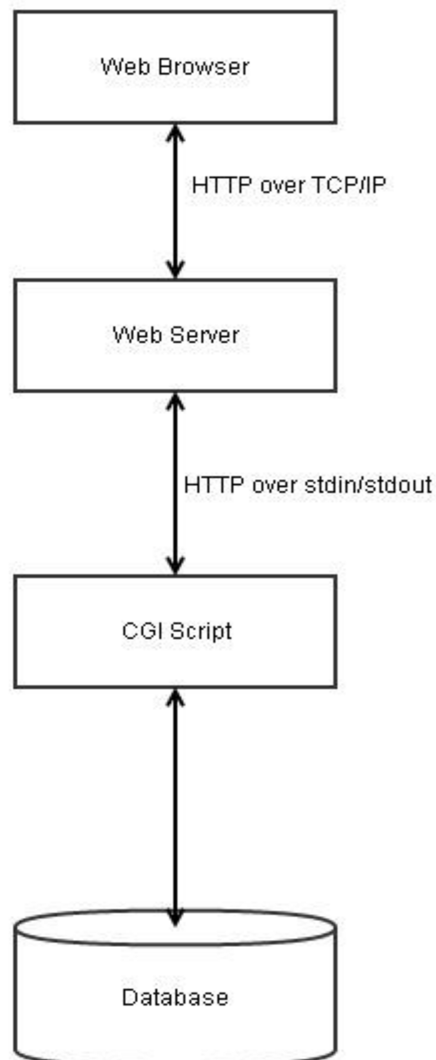

Figure 5.1: CGI Architecture

The steps involved in the execution of the program are as follows [45]:

1. Provide the URL of the program file (.py extension) to the web browser.
2. Web browser contacts the web server in order to fetch the program file.
3. URL will then be parsed by HTTP server to check whether the file exists. If file does not exist, server will give an error. If file exists, then python interpreter will be invoked which runs the script. Script reads the HTTP request data from stdin and sends back the output of the program to the web browser via stdout.

5.2 Snapshots

Figure 5.2 shows the first page that we get when we enter the URL of the program file in the web browser. It is being made user interactive by using the form asking for user inputs. User needs to input the pattern to be matched, the path of the file where DNA sequence is stored, minimum and maximum value of global length constraints, option whether the user wants the output with one-off constraint or without one-off constraint.

After entering the above information, user needs to click on the calculate button in order to get the result.

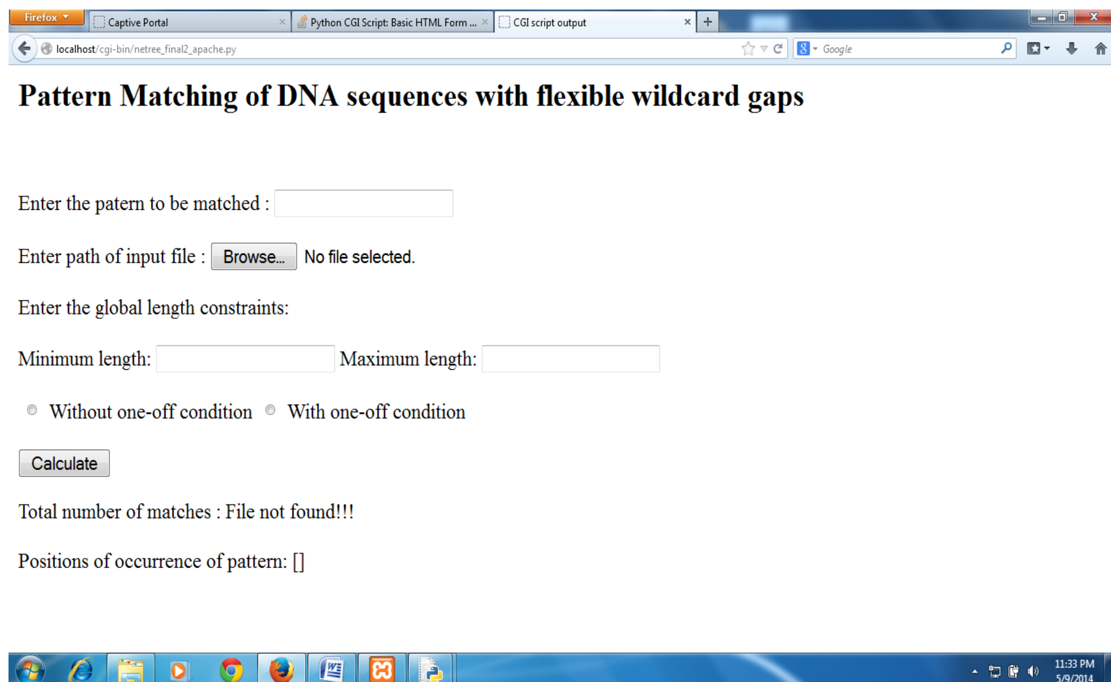

Figure 5.2: First Page

Figure 5.3 shows the content of the input file. Input file consists of the DNA sequence. While reading the DNA Sequence, spaces and newline characters are being removed by the program.

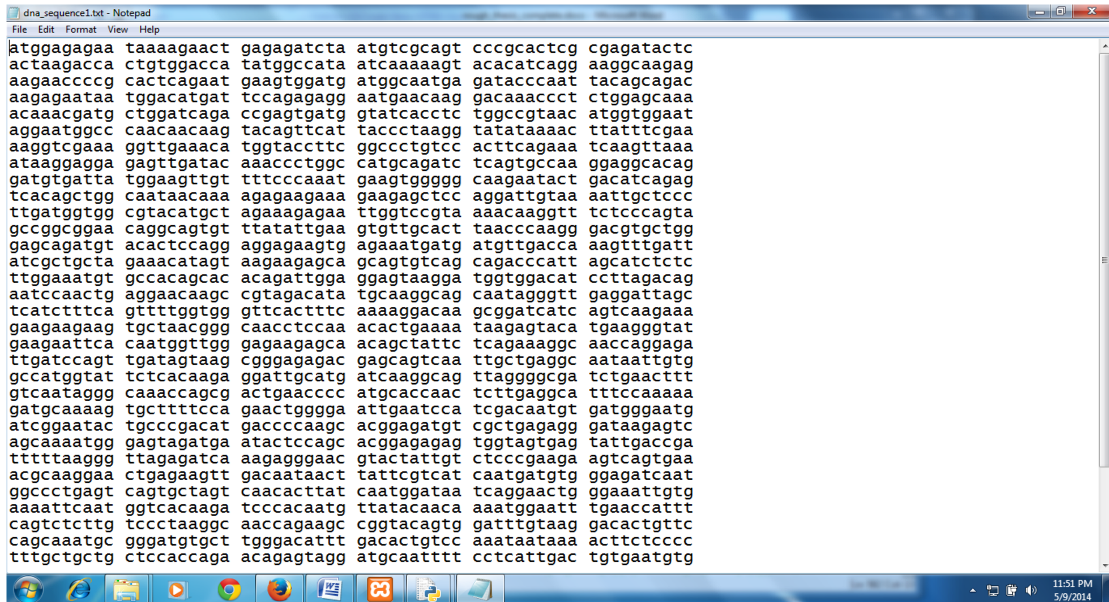

Figure 5.3: Input File containing the DNA sequence

Figure 5.4 shows the output of the program when wrong pattern is being entered by the user. Here by mistake user entered the pattern containing character other than in the alphabet {a, t, c, g}.

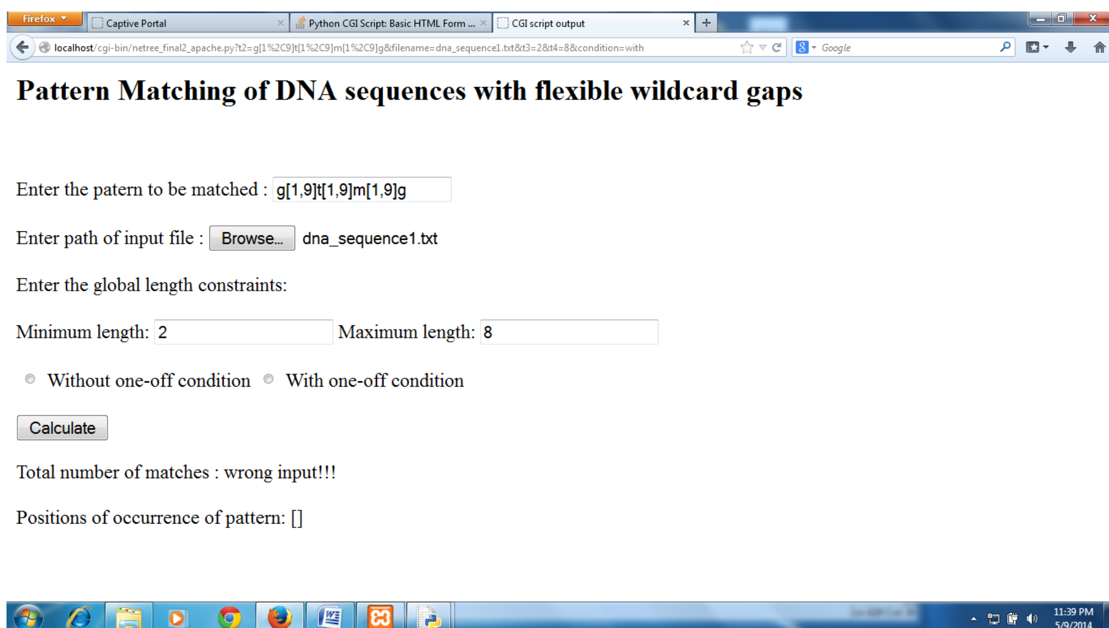

Figure 5.4: Output when wrong pattern is entered by the user

Figure 5.5 shows the output when user forgets to upload the file containing DNA sequence. It gives an error “File not found!!!”

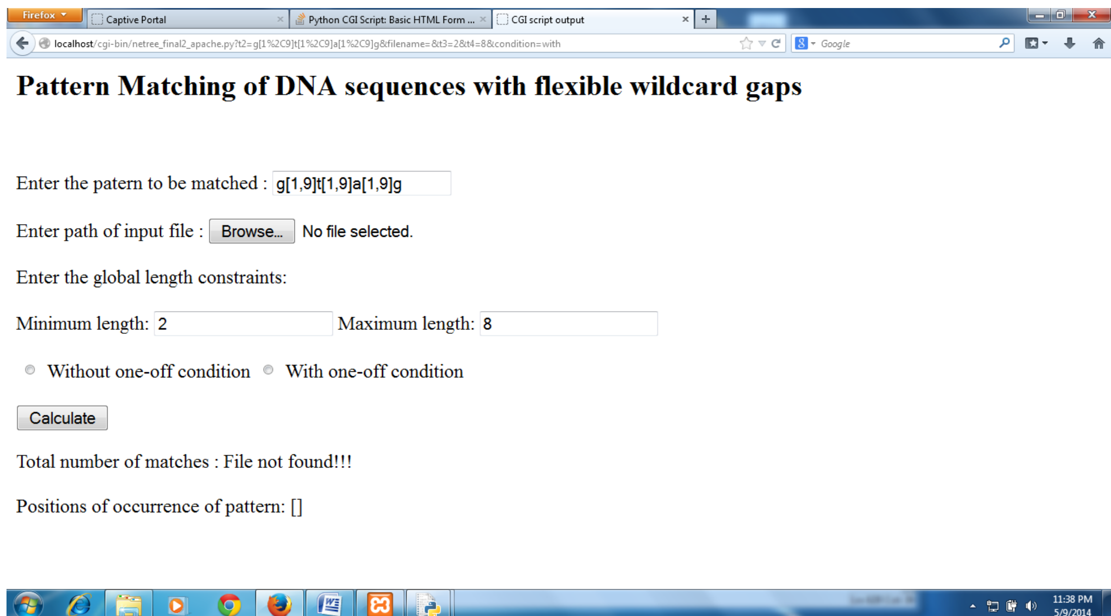

Figure 5.5: Output when path of file is not given

Figure 5.6 shows the output of the program when user enters an invalid range of global length constraints. Invalid range here implies that the minimum value of global length constraint is greater than the maximum value of global length constraint as provided by user.

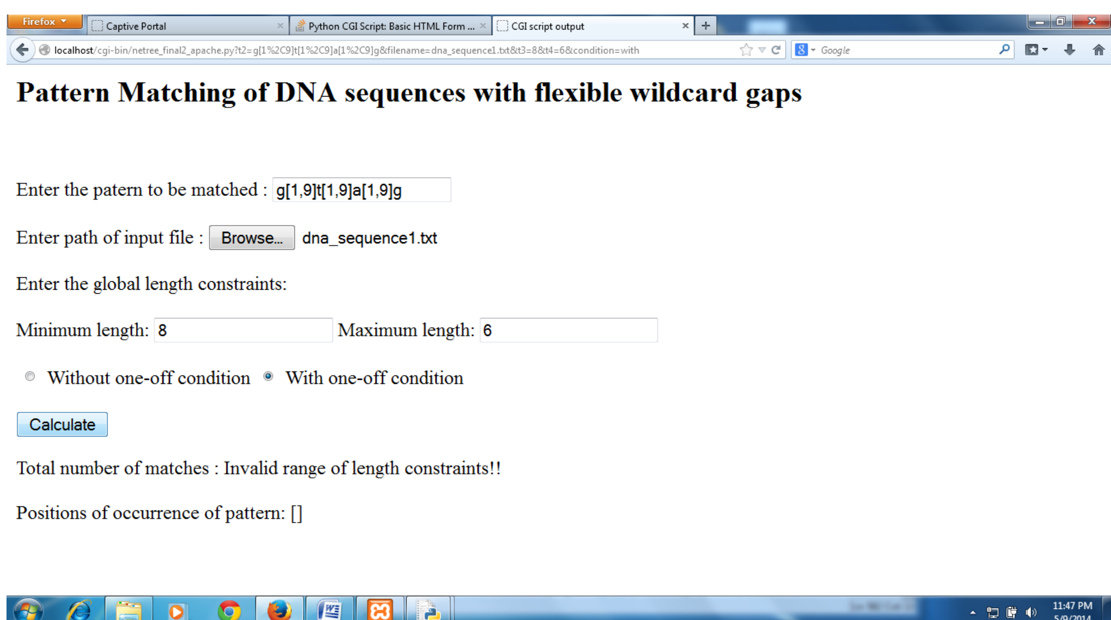

Figure 5.6: Output when invalid range of global length constraints is provided

The output of the program for the given pattern and sequence when the user selects the “Without one-off” option is shown in Figure 5.7.

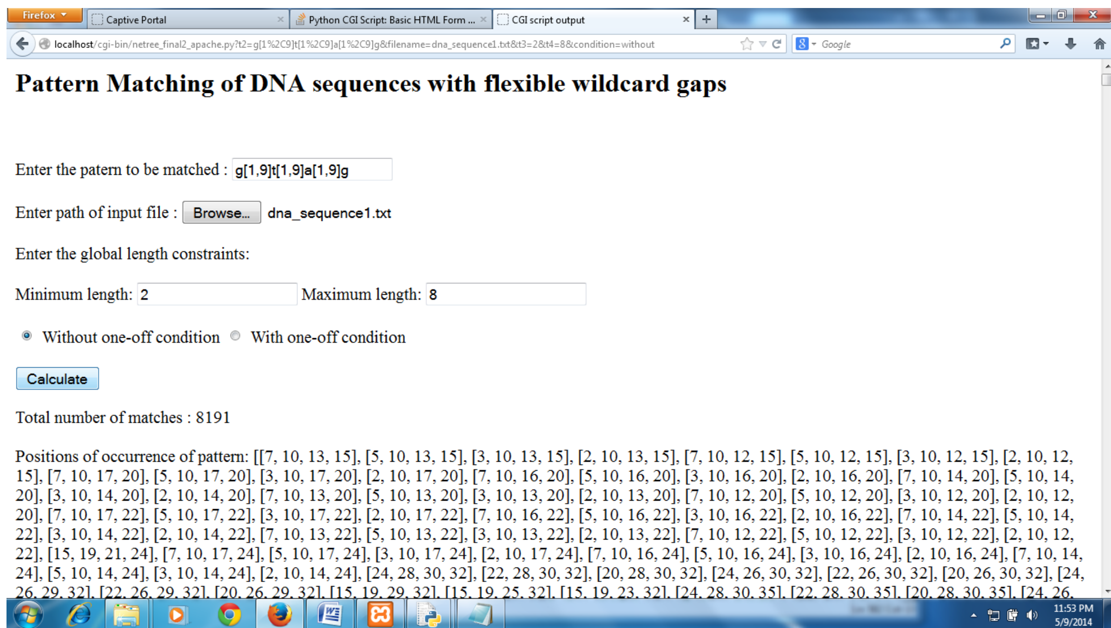

Figure 5.7: Output without one-off condition

The output of the program for the given pattern and sequence when the user selects the “with one-off” option is shown in Figure 5.8.

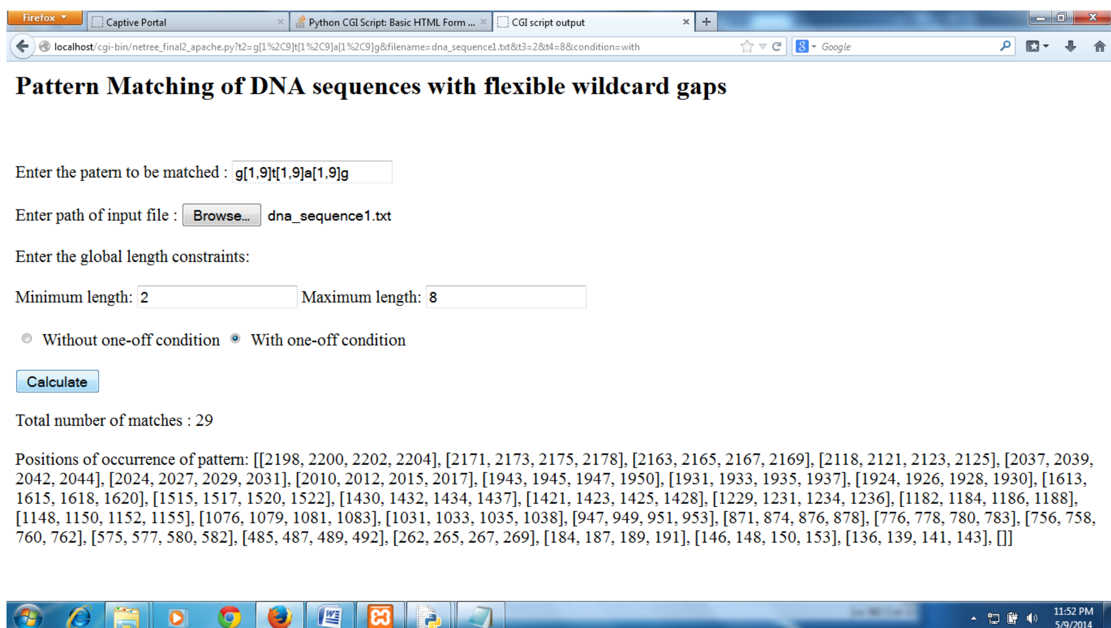

Figure 5.8: Output with one-off condition

5.3 Experimental Results

5.3.1 Experiment on Real Data

MOGO has been tested on real world biological data. 8 different segments of the H1N1 (Swine Flu) virus are downloaded from the website of National Center for Biotechnology Information [46]. Locus of the sequences and their length are given in the Table 5.1. Each of these 8 biological sequences has been tested against 4 different patterns with wildcard gaps given by Min et al. [37]. These 4 patterns are shown in Table 5.2. Table 5.3 specifies the minimum and maximum length parameters of global length constraint for each of the pattern given in Table 5.2.

Table 5.1: Biological Sequences

Sequence Number	Locus	Length
S1	CY058563	2286
S2	CY058562	2299
S3	CY058561	2169
S4	CY058556	1720
S5	CY058559	1516
S6	CY058558	1418
S7	CY058557	982
S8	CY058560	844

Table 5.2: Patterns with wildcard gaps

Pattern Number	Pattern
P1	a[0,3]t[0,3]a[0,3]t[0,3]a[0,3]t[0,3]a[0,3]t[0,3]a[0,3]t[0,3]a
P2	g[1,5]t[0,6]a[2,7]g[3,9]t[2,5]a[4,9]g[1,8]t[2,9]a
P3	g[1,9]t[1,9]a[1,9] g[1,9]t[1,9]a[1,9] g[1,9]t[1,9]a[1,9]g[1,9]t
P4	g[1,5]t[0,6]a[2,7]g[3,9]t[2,5]a[4,9]g[1,8]t[2,9]a[1,9]g[1,9]t

Table 5.3: Global Length Constraints

Pattern Number	Minimum Length	Maximum Length
P1	11	41
P2	24	57
P3	21	101
P4	27	73

Table 5.4 shows the results obtained by conducting the experiment and its comparison with the results of the existing algorithms SAIL and HSO as given in [41]. According to the results, MOGO gives 43.75% better results than the algorithm SAIL and 37.50% better results than the algorithm HSO. Hence MOGO performs better than the existing algorithms SAIL and HSO by searching more number of occurrences of a pattern in a sequence.

Table 5.4: Experimental Results

Pattern	Algorithm	S1	S2	S3	S4	S5	S6	S7	S8
P1	SAIL	13	9	10	15	11	5	3	3
	HSO	13	9	10	15	11	5	3	3
	MOGO	13	9	10	15	11	5	3	3
P2	SAIL	66	69	59	54	42	39	31	27
	HSO	67	71	62	54	42	41	33	28
	MOGO	67	73	65	55	44	44	33	32
P3	SAIL	66	69	66	54	45	42	33	28
	HSO	64	70	68	52	43	43	33	26
	MOGO	68	70	72	52	44	43	32	27
P4	SAIL	49	50	49	40	32	31	24	20
	HSO	51	58	52	46	37	30	26	21
	MOGO	48	58	54	48	37	35	26	22

5.3.2 Experiment on Artificial Data

For the problem taken into consideration, no complete solution has been developed so far. In this subsection, we analyze the effect of different Constraints on the Completeness of the Solution. Some of the parameters that affects the completeness of the solution includes length of the pattern i.e. m and the maximum wildcard gap between consecutive letters of the pattern i.e. W. For the purpose of this analysis, artificial data is used as it is having the complete solution for the problem and is downloaded from [47]. Artificial data is generated by data generator [43]. Input to the data generator is alphabet Σ, pattern, length of sequence n and maximal support *sup* and output is the sequence with exact support value of pattern in sequence under the one-off condition. So in this way, with artificial data, we have complete solution of the problem.

A parameter, Accuracy, is used to measure the completeness of the algorithm. Accuracy is defined as:

$$Accuracy = \frac{\text{Num_occ}}{Total_occ}$$

where *Num_occ* is number of occurrences of pattern returned by the algorithm MOGO and *Total_occ* is the total number of all possible occurrences of the pattern in the sequence.

We get *Total_occ* from the artificial data set. For the same combination of pattern and sequence in artificial data set, we run the algorithm MOGO in order to obtain *Num_occ* .

Figure 5.9 shows the effect of length of the pattern *m* on accuracy of the algorithm. Analysis is done for 13 different values of m ranging from 2 to 14. For each value of m, 10 artificial data sets are considered and the accuracy for that particular value of m is equal to the average of accuracies corresponding to those 10 artificial data sets. From the graph, it is clearly visible that with the increase in m i.e. length of the pattern, accuracy of the algorithm is gradually decreasing.

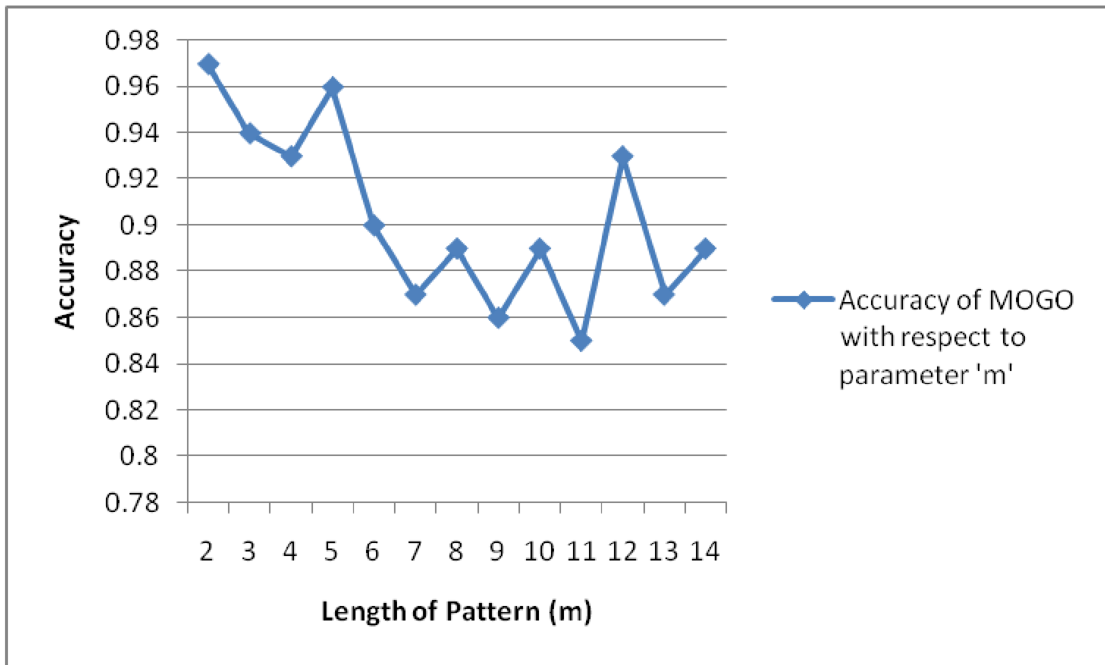

Figure 5.9: Effect of length of the pattern on the accuracy of the algorithm MOGO

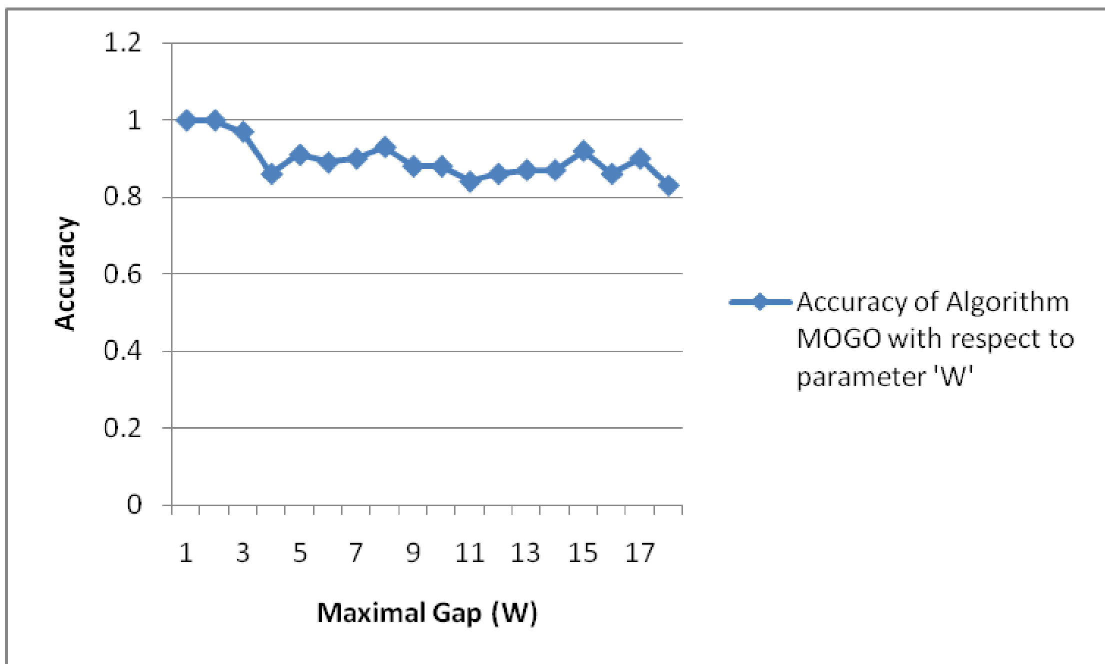

Figure 5.10: Effect of maximum wildcard gap on the accuracy of the algorithm MOGO

Figure 5.10 shows the effect of maximum wildcard gap between consecutive letters of the pattern W on accuracy of the algorithm. Analysis is done for 18 different values of W ranging from 1 to 18. For each value of W, 10 artificial data sets are considered and the accuracy for that particular value of W is equal to the average of accuracies corresponding to those 10 artificial data sets. From the graph, it is clearly visible that with the increase in W i.e. gap, accuracy of the algorithm is gradually decreasing. As the gap increases, the probability of overlapping of the possible occurrences becomes higher. Thus there are more chances of losing the occurrences leading to loss of accuracy.

CHAPTER – 6
CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

In this thesis, we considered the problem of pattern matching with flexible wildcard gaps between every two consecutive letters of pattern under the one-off constraint. This problem adds more complexity and flexibility to the traditional pattern matching. All the potential applications of the mentioned problem have been listed and its significance in the field of bioinformatics has been studied in detail.

Algorithms for traditional pattern matching have been briefly explained. Different algorithms based on greedy approaches to solve the problem of pattern matching with flexible wildcard gaps between every two consecutive letters of pattern under the one-off constraint have been studied in detail along with their pros and cons. Through this study, the effect of adding different constraints to the traditional pattern matching problem and thus leading to difficulty in achieving optimal solution for the problem is understood. Comparative analysis of these algorithms has been done on the basis of their complexities, data structure used by them and matching strategies incorporated in these algorithms.

We then proposed the new algorithm, MOGO (Maximum Occurrences with Global length constraints and One-off condition) based on the Nettree data structure which performs better than its peers SAIL and HSO according to theoretical analysis and experimental results. To show the elaborate working of this algorithm, an illustration example has been provided. MOGO performs better by giving more number of pattern matches in a real world biological sequence. MOGO is based on a heuristic technique and thus doesn't provide complete solution to the problem. The time and space complexity of MOGO is O(W*n*(n+m)) and O(W*m*n) respectively, where m is the length of the pattern, n is the length of the sequence and W is the maximum gap between consecutive letters of the pattern.

6.2 Future Scope

In the future, this work can be extended to allow for some errors while matching the pattern i.e. approximate pattern matching.

The approach suggested in this thesis can be used to support Multi-pattern matching with flexible wildcard gaps under one-off condition. Multi-pattern matching refers to matching more than one pattern simultaneously against the given biological sequence.

Taking into consideration the given constraints, the technique used to match the patterns against biological sequences is applied on the "Nettree data structure". Applying the proposed approach on different data structure might give better result.

Different technique can be applied for pattern matching on the same data structure used in this work in order to reduce the time complexity and improve the results.

REFERENCES

[1] G. Navarro, “Flexible pattern matching,” in Journal of Applied Statistics. Citeseer, 2002.

[2] S. Neuburger, “Pattern matching algorithms: An overview,” 2009.

[3] R. Bhukya and D. Somayajulu, “Exact multiple pattern matching algorithm using dna sequence and pattern pair.” International Journal of Computer Applications, vol. 17, 2011.

[4] S. Wu and U. Manber, “Fast text searching: allowing errors,” Communications of the ACM, vol. 35, no. 10, pp. 83–91, 1992.

[5] “Princeton University website, Department of Computer Science,” http://www.cs.princeton.edu/~rs/AlgsDS07/21PatternMatching.pdf, [Online; accessed 06-May-2014].

[6] M. Zhang, B. Kao, D. W. Cheung, and K. Y. Yip, “Mining periodic patterns with gap requirement from sequences,” ACM Transactions on Knowledge Discovery from Data (TKDD), vol. 1, no. 2, p. 7, 2007.

[7] “Course Web Service for University of Washington Computer Science and Engineering,”http://courses.cs.washington.edu/courses/cse527/00wi/lectures/roottr.pdf, [Online; accessed 06-May-2014].

[8] “University of Illinois at Chicago website,” http://www.uic.edu/classes/phys/phys461/phys450/ANJUM04/, [Online; accessed 06-May-2014].

[9] “Wikipedia,” http://en.wikipedia.org/wiki/Nucleicacidsequence; [Online; accessed 06-May-2014].

[10] Y.-I. Chang, J.-R. Chen, and M.-T. Hsu, “A hash trie filter method for approximate string matching in genomic databases,” Applied Intelligence, vol. 33, no. 1, pp. 21–38, 2010.

[11] C. L. Lam, “Approximate string matching in dna sequences,” Ph.D. dissertation, The University of Hong Kong, 2003.

[12] “Personal Web pages of students, faculty and staff of Harvard University,” http://www.people.fas.harvard.edu/~ junliu/sequence_analysis.pdf, [Online; accessed 06 – May-2014]

[13] “A dictionary of computer related terms,” http://www.techterms.com/definition/wildcard, [Online; accessed 06-May-2014].

[14] N. Pisanti, M. Crochemore, R. Grossi, and M.-F. Sagot, “Bases of motifs for generating repeated patterns with wild cards,” Computational Biology and Bioinformatics, IEEE/ACM Transactions on, vol. 2, no. 1, pp. 40–50, 2005.

[15] R. S. Ayg¨un, “S2s: structural-to-syntactic matching similar documents,” Knowledge and information systems, vol. 16, no. 3, pp. 303–329, 2008.

[16] R. Cole, L.-A. Gottlieb, and M. Lewenstein, “Dictionary matching and indexing with errors and don’t cares,” in Proceedings of the thirty-sixth annual ACM symposium on Theory of computing. ACM, 2004, pp. 91–100.

[17] U. Manber and R. Baeza-Yates, “An algorithm for string matching with a sequence of don’t cares,” Information Processing Letters, vol. 37, no. 3, pp. 133–136, 1991.

[18] T. Akutsu, “Approximate string matching with variable length don’t care characters,” IEICE TRANSACTIONS ON INFORMATION AND SYSTEMS E SERIES D, vol. 79, pp. 1353–1354, 1996.

[19] H. Wang, T. Xiang, and X. Hu, “Research on pattern matching with wildcards and length constraints: Methods and completeness,” 2012.

[20] G. Chen, X. Wu, X. Zhu, A. N. Arslan, and Y. He, "Efficient string matching with wildcards and length constraints," Knowledge and information systems, vol. 10, no. 4, pp. 399–419, 2006.

[21] G. Navarro and M. Raffinot, Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences. Cambridge University Press, 2002.

[22] "Washington State University, School of Electrical Engineering and Computer Science," http://www.eecs.wsu.edu/~ananth/CptS317/Lectures/FiniteAutomata.pdf, [Online; accessed 06-May-2014].

[23] Y. Liu, X. Wu, X. Hu, J. Gao, and C. Wang, "Pattern matching with wildcards based on multiple suffix trees," in Granular Computing (GrC), 2012 IEEE International Conference on. IEEE, 2012, pp. 320– 325.

[24] "University of Central Florida website," http://www.cs.ucf.edu/shzhang/Combio11/lec3.pdf, [Online; accessed 06-May-2014].

[25] D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt, "Fast pattern matching in strings," SIAM journal on computing, vol. 6, no. 2, pp. 323–350, 1977.

[26] R. Baeza-Yates and G. H. Gonnet, "A new approach to text searching," Communications of the ACM, vol. 35, no. 10, pp. 74–82, 1992.

[27] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," Communications of the ACM, vol. 20, no. 10, pp. 762–772, 1977.

[28] R. N. Horspool, "Practical fast searching in strings," Software: Practice and Experience, vol. 10, no. 6, pp. 501–506, 1980.

[29] M. Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter, "Speeding up two string-matching algorithms," Algorithmica, vol. 12, no. 4-5, pp. 247–267, 1994.

[30] G. Navarro and M. Raffinot, "Fast and flexible string matching by combining bit-parallelism and suffix automata," Journal of Experimental Algorithmics (JEA), vol. 5, p. 4, 2000.

[31] C. Allauzen, M. Crochemore, and M. Raffinot, "Efficient experimental string matching by weak factor recognition*," in Combinatorial Pattern Matching. Springer, 2006, pp. 51–72.

[32] M. J. Fischer and M. S. Paterson, "String-matching and other products." DTIC Document, Tech. Rep., 1974.

[33] S. Muthukrishnan and K. Palem, "Non-standard stringology: Algorithms and complexity," in Proceedings of the twenty-sixth annual ACM symposium on Theory of computing. ACM, 1994, pp. 770–779.

[34] P. Indyk, "Faster algorithms for string matching problems: Matching the convolution bound," in Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on. IEEE, 1998, pp. 166–173.

[35] A. Kalai, "Efficient pattern-matching with don't cares," in Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2002, pp. 655–656.

[36] G. Navarro and M. Raffinot, "Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching," Journal of Computational Biology, vol. 10, no. 6, pp. 903–923, 2003.

[37] F. Min, X. Wu, and Z. Lu, "Pattern matching with independent wildcard gaps," in Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on. IEEE, 2009, pp. 194–199.

[38] H. Wang, F. Xie, X. Hu, P. Li, and X. Wu, "Pattern matching with flexible wildcards and recurring characters," in Granular Computing (GrC), 2010 IEEE International Conference on. IEEE, 2010, pp. 782– 786.

[39] D. Guo, X.-L. Hong, X.-G. Hu, J. Gao, Y.-L. Liu, G.-Q. Wu, and X. Wu, “A bit-parallel algorithm for sequential pattern matching with wildcards,” Cybernetics and Systems, vol. 42, no. 6, pp. 382–401, 2011.

[40] J. Qiang, W. Tian, D. Guo, and X. Wu, “Online pattern matching with wildcards,” in Granular Computing (GrC), 2012 IEEE International Conference on. IEEE, 2012, pp. 394–399.

[41] Y. Wu, X. Wu, H. Jiang, and F. Min, “A nettree for approximate maximal pattern matching with gaps and one-off constraint,” in Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on, vol. 2. IEEE, 2010, pp. 38–41.

[42] Y. Wu, X. Wu, F. Min, and Y. Li, “A nettree for pattern matching with flexible wildcard constraints,” in Information Reuse and Integration (IRI), 2010 IEEE International Conference on. IEEE, 2010, pp. 109–114.

[43] D. Guo, X. Hu, F. Xie, and X. Wu, “Pattern matching with wildcards and gap-length constraints based on a centrality-degree graph,” Applied intelligence, vol. 39, no. 1, pp. 57–74, 2013.

[44] D. He, X. Wu, and X. Zhu, “Sail-approx: an efficient on-line algorithm for approximate pattern matching with wildcards and length constraints,” in Bioinformatics and Biomedicine, 2007. BIBM 2007. IEEE International Conference on. IEEE, 2007, pp. 151–158.

[45] “tutorialspoint (TP) Simply Easy Learning website” http://www.tutorialspoint.com/python/python_cgi_programming.htm, [Online; accessed 06 -May - 2014].

[46] “National center for biotechnology information website,” http://www.ncbi.nlm.nih.gov/, [Online; accessed 06-December-2013].

[47] “HFUT Data Mining and Intelligent Computing Laboratory Website,” http://dmic.hfut.edu.cn/HFUT_DMIC/DanGuo/test/, [Online; accessed 10-May-2014].

LIST OF PUBLICATIONS

[1] Anu Dahiya and Deepak Garg, "Maximal Pattern Matching with Flexible Wildcard Gaps and One-off Constraint," 3rd International Conference on Advances in Computing, Communications and Informatics, IEEE, 2014. [Accepted]