Mobile Applications and Algorithms for Information Security

Alshammary, Miznah

Awarding institution:
King's College London

Mobile Applications and Algorithms for Information Security

Miznah Hizam AlShammary
Department of Informatics King's College London

This dissertation is submitted for the degree of
Doctor of Philosophy

King's College London
September 2020

Supervisors

First Supervisor

Prof. Costas S. Iliopoulos, King's College London.

Second Supervisor

Dr Grigorios Loukides, King's College London.

Examiners

Prof. Frantisek Franek

McMaster University, Hamilton, Ontario, Canada.

Dr Fiaz Hussain

Cardiff Metropolitan University, Cardiff, United Kingdom.

I, Miznah AlShammary

a proud member of my family, dedicate this thesis to my parents . . .

Hizam Mohammed AlAslam AlShammary, God bless his soul

Latifah Ahamed AlWadani AlDossari, God Save her

And my family members

Mohammed, Sarah, Nouf, and Sultan.

Declaration of Authorship

I hereby declare that except when specifically referencing the work of others, this dissertation's contents are original and have not been submitted in whole or in part for any other qualifications at this, or any other university.
I declare that this thesis, titled "Mobile Applications and Algorithms for Information Security" and the works presented in it are my own; some work has been conducted in collaboration with others, as detailed in the text and Acknowledgements.

Miznah Hizam AlShammary
September 2020

Acknowledgements

I'd like to express my deepest appreciation to who helped me achieve my PhD.

My supervisors [1][2], for their direction, encouragement, counsel, and support during my studies. To our research group, and to all my published research co-authors for their excellent collaboration. To department of informatics and its professional services team for their expert administrative help.

My thesis examiners [1][2], for consenting to evaluate my dissertation and for their helpful recommendations and amendments.

To the most supportive person in my life, my mother, Latifah, who set me on the road to this PhD a long time ago, and to my late father, Hizam, who instilled in me love of learning. To my sister, Sarah, who supported me whilst undergoing her own MSc journey in London. And to my brother, Mohammed, and his family, Nouf and Sultan, for their unlimited support, love, and understanding.

To my best friends Nouf Alzeer and Shams Alshumaysi who were there for me with constant support, and with continuous encouragement.

Finally, I am very grateful to my beloved country, Kingdom of Saudi Arabia, for providing the full scholarship that allowed me to conduct this research and the opportunity to attend conferences and meet so many interesting people. I give my thanks to Imam Abdul Rahman bin Faisal University in Dammam for having such faith in me and for the scholarship as one of their faculty members, which led me to succeed in my career.

Abstract

The relevance of bioinformatics technology has grown dramatically over the last decade due to the fast change in human-centric technologies. Bioinformatics now plays a fundamental role in many significant sectors of everyday life, including health, digital forensics, and applications and digital asset security, by providing authentication and authorisation based on people's unique biometric traits. When assisting technology and information systems, bioinformatics posed queries such as "Why," "What if," and "When," among others. Bioinformatics opens us to a universe of possibilities in various real-life applications. However, bioinformatics comes with its own set of difficulties. The security of bioinformatics algorithms and applications has been a significant source of worry in recent years. On the opposite side of the pillar, the cybercrime arena is expanding daily as current information and cyber security tactics progress. This dissertation aims to investigate, analyse, and resolve information and cyber security challenges arising from typical bioinformatics techniques and their algorithms and compute applications. For calculating cyclic regularities of strings, many algorithms and applications are presented. This dissertation examines privacy and security challenges with structural regularities in strings such as periods and covers, among other things.

Publications

Cited in thesis:

Chapter 2:

Ajala, O. I., Alshammary, M. H., Alzamel, M. A. M., Gao, J., Iliopoulos, C., Radoszewski, J., Rytter, W. & Watson, B., 15 May 2019. On the Cyclic Regularities of Strings. *Artificial Intelligence Applications and Innovations - AIAI 2019 IFIP WG 12.5 International Workshops: MHDW and 5G-PINE 2019, Proceedings. Maglogiannis, I., Iliadis, L., MacIntyre, J. & Pimenidis, E. (eds.). Springer*, Vol. 560. pp. 219-224 (IFIP Advances in Information and Communication Technology, vol. 560). Springer, Cham. https://doi.org/10.1007/978-3-030-19909-8_19

Chapter 3:

Alshammary M.H., Iliopoulos C.S., Khan M.R. (2020) Fingerprints Recognition System-Based on Mobile Device Identification Using Circular String Pattern Matching Techniques. In: Maglogiannis I., Iliadis L., Pimenidis E. (eds) Artificial Intelligence Applications and Innovations. AIAI 2020 IFIP WG 12.5 International Workshops. AIAI 2020. IFIP Advances in Information and Communication Technology, vol 585. Springer, Cham. https://doi.org/10.1007/978-3-030-49190-1_20.

Chapter 4:

Alshammary M.H., Iliopoulos C.S., Mohamed M., Vayani F. (2020) Application and Algorithm: Maximal Motif Discovery for Biological Data in a Sliding Window. In: Maglogiannis I., Iliadis L., Pimenidis E. (eds) Artificial Intelligence Applications and Innovations. AIAI 2020 IFIP WG 12.5 International Workshops. AIAI 2020. IFIP Advances in Information and Communication Technology, vol 585. Springer, Cham. https://doi.org/10.1007/978-3-030-49190-1_19.

Not cited in thesis:

Alshammary M., Alzamel M., Iliopoulos C., Watson R.E., Watson B.W. (2019) A Brief Overview of Dead-Zone Pattern Matching Algorithms. In: MacIntyre J., Maglogiannis I., Iliadis L., Pimenidis E. (eds) Artificial Intelligence Applications and Innovations. AIAI 2019. IFIP Advances in Information and Communication Technology, vol 560. Springer, Cham. https://doi.org/10.1007/978-3-030-19909-8_18

M. H. Alshammary and F. A. Alanezi, "A Review of Recent Developments in NOMA & SCMA Schemes for 5G Technology," *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*, Anyang, 2017, pp. 55-59, doi: 10.1109/DCABES.2017.19. https://ieeexplore.ieee.org/document/8253035

F. Alanezi and M. H. Alshammary, "Review of Social Networking Applications for Infectious Disease Management Systems in Saudi Arabia: Current Status & Future Prospects," *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*, Anyang, 2017, pp. 100-104, doi: 10.1109/DCABES.2017.29. https://ieeexplore.ieee.org/document/8253045

Java the language of the era Book, M. H. Alshammary, 2014 King Fahd National Library, 1435/4667, 978-603-8093-85-6.

Table of Contents

List of Figures

List of Tables

1 Introduction

Numerous effective bioinformatic algorithms are investigated in this thesis to aid in the resolution of several security and privacy challenges, such as biometrics and big data. The thesis's significant body comprises various chapters based on previously published works. These are followed by a series of concepts necessary to comprehend the purpose of studying the cyclic factors of strings in Chapter 2. I have developed algorithms and implemented software applications on a circular string pattern for biological data. My research shifted to information security while I worked on these software applications. I addressed those security concerns by offering solutions from both an algorithmic and a practical standpoint. Chapter 3 presented algorithms and implementations for fingerprint identification on mobile devices utilising circular string pattern matching. Because circular string (matching and discovery) research is still in its infancy from a biological standpoint, few efficient software programmes are available for biologists to handle circular string-matching problems. Chapter 4 implemented and solved maximal motif discovery challenges using circular pattern discovery for sliding windows. Chapter 5 summarises the findings of previous chapters, from which we make conclusions and plan future studies.

This dissertation comprises of a series of algorithms and implementations (based on string specific algorithms, data-structures, and properties) that have been extensively studied in Chapter 2 and used to tackle many key security research challenges that find direct or indirect applications in genomic data analysis in Chapter 3 and 4.

The solutions provided in this thesis will gain strength and empowerment by looking at similar challenges from diverse perspectives and combining many fields to tackle similar difficulties. And that is a significant aspect in this thesis, which ranges from detecting and analysing genomic data using next-generation sequencing for biological data processing to fingerprint matching approach utilising circular string-matching algorithm to tackle the human virus issue in bioinformatics. where the overall thesis system diagram (As shown in Figure 1-1) shows this overlap between the domains of information security and bioinformatics is equally helpful to both fields of study. It integrates more than one file to fix any concerns, giving the solutions strength and empowering them by looking at comparable challenges from other angles.

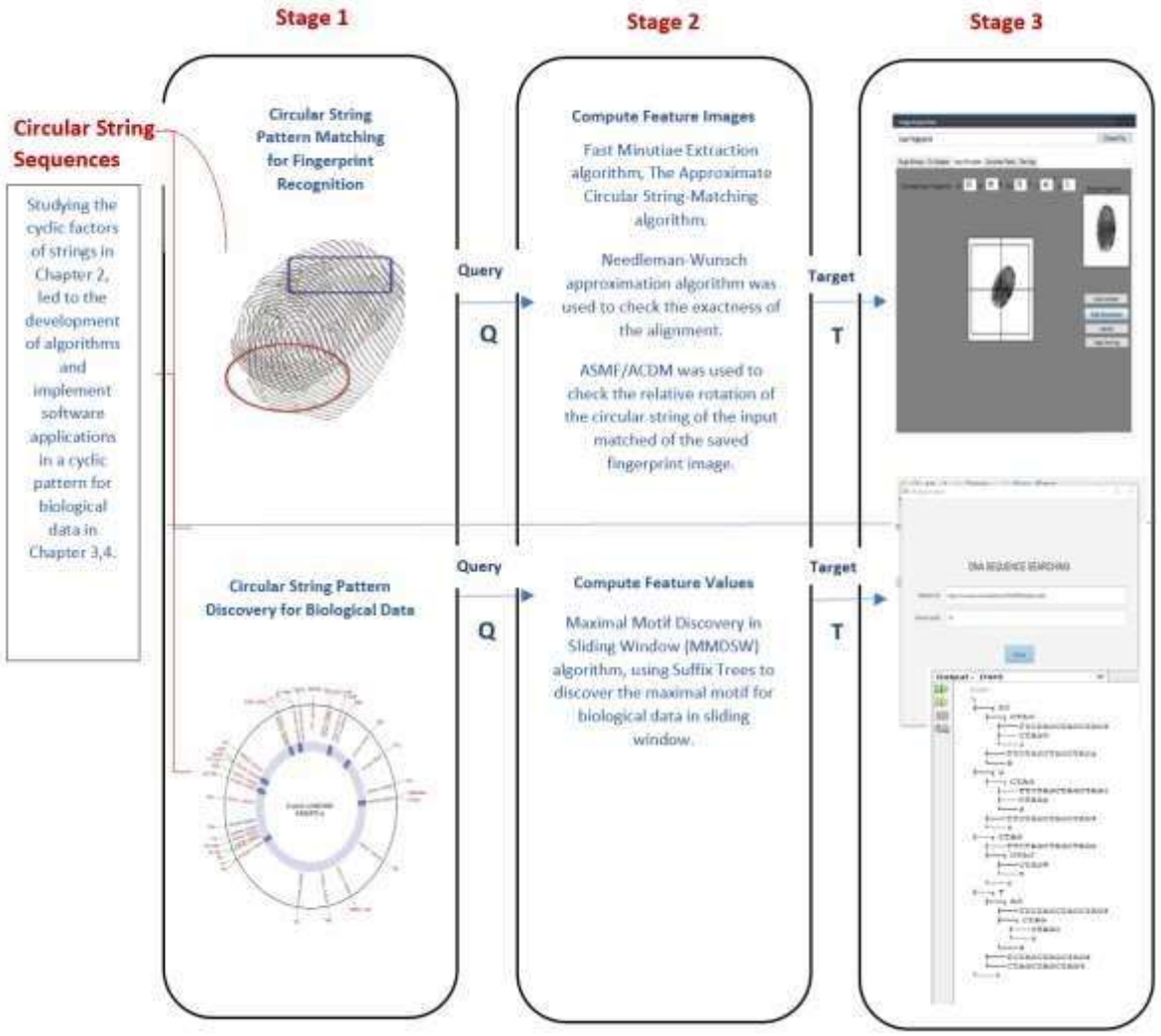

Figure 1-1 Overview of the overall thesis system

1.1 Bioinformatics

Bioinformatics is defined as an inter-disciplined field that aims to understand large and complex biological data by developing software tools and methods (Baxevanis, Bader, & Wishart, 2020). Bioinformatics can be categorised as a sub-discipline of biology and information technology or computer science mainly concerned with collecting, storing, analysing, and disseminating biological data with the help of efficient computer programs or algorithms. If computer and information technology are associated with any discipline, information and cyber security risks are autonomously generated due to the vast cybercrime space. With its uprising in different realworld applications such as forensics, health, research and validations, bioinformatics also brings significant security risks. Therefore, a prominent intersection can be seen between bioinformatics and information security.

The bioinformatics algorithms and applications can accept such susceptible data. Biometric data, for example, is analysed using string or stringology methods; difficulties in biological research, for example, may be by handled by develop algorithms and implement software applications for analysing large data sequences in information security such as fingerprints and genetic data (molecular sequencing) in Chapters 3 and 4. So, if the volume of data is large, the consistency of the received information improves. Still, there is a more incredible computing difficulty in the analysis and processing simultaneously. Though computer capacity has increased dramatically over the last two decades, the rate at which data is generated has outpaced how calculations are processed. As a result, corporations have developed feasible solutions to the challenges encountered during data interpretation to solve a few privacy concerns and assist/improve information security research. Because of the efficiency of algorithmic tools, this is now achievable. As a result, the proposed algorithms in this thesis were implemented using open-source software that may be immediately plugged into mobile devices and utilised by future researchers. My thesis focused on information security when these algorithms and implementations were developed and deployed.

To summarise, this intersection between information security and bioinformatics may assist both sectors of research projects. Furthermore, more than one file is integrated to reinforce the answers and address the problems by looking at similar problems from different angles.

1.2 Mobile Security

Due to their general and operational natures, mobile applications are always vulnerable to cybersecurity attacks. Cybercrimes and the increased incidents of identity theft have revealed the significance of information security. Cyber-attacks, i.e. identity theft, can quickly and recurrently be performed on targeted devices and applications without proper information security measures. For example, the concerned individuals or groups reportedly carry out regular identification and verification-related activities through biometrics for mobile devices (fingerprints). As a result, concerns with fingerprint matching will be discussed in Chapter 3. Furthermore, a proposed solution will be explored using a fast minutiae extraction algorithm for approximate circular string matching to solve the drawbacks of earlier solutions. Smartphones, laptops, and Personal Digit Assistants (PDAs) have become an integral part of everyone's life nowadays. These digital gadgets serve as a bridge between the personal life of an individual and business transactions. Due to their multiple functionalities, in Great Britain, these devices' use increased from 57% in 2006 to 89% in 2016 and still increasing day by day beyond 2016 (As shown in Figure 1-2).

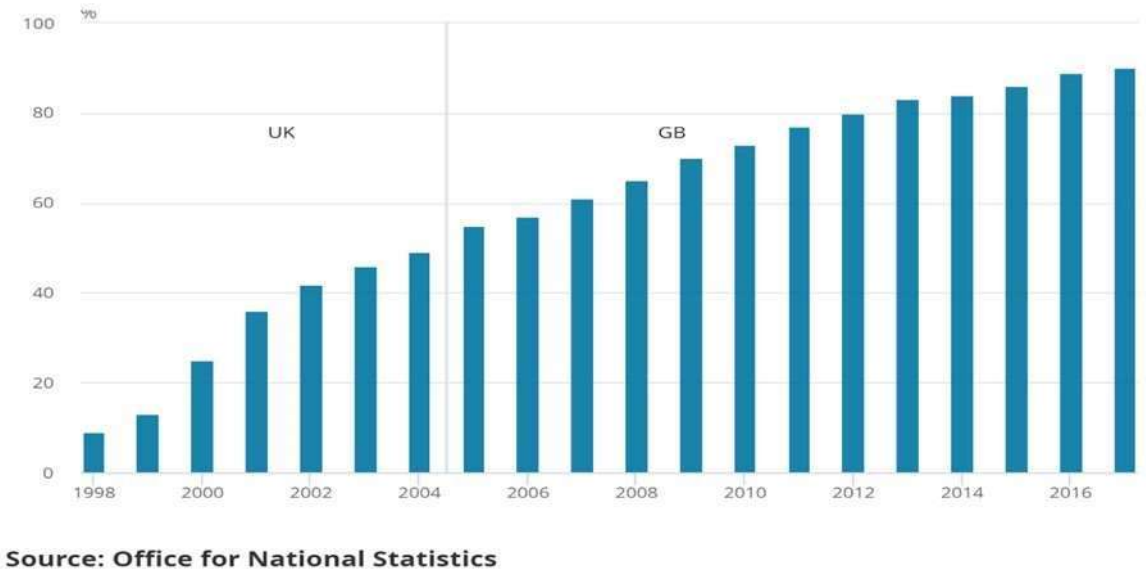

Figure 1-2 Internet access growth in the office for National Statistics databases. Data retrieved from the Office for National Statistics. www.ons.gov.uk

Instead of a heavy computer, a smartphone is the first preference of people in this day and age, since it is a portable gadget and a pocket or handbag can easily accommodate it. Across the workplaces, the use of PDAs and mobile phones is one of the critical factors to improving communication. Besides giving personal information to the owner, the operations of these devices are nearly identical to those of desktops. Features like notepad, phonebook and calendar are present in them. Mobile devices receive text messages and voice calls along with these features. Because of their growing demand, activities such as theft and hacking have also increased.

Consequently, the adoption of security measures is significant. Organisations use these devices for data storage. Their valuable information and data could be lost due to a lack of security; hence, their output is compromised. Resultantly, enhanced security measures for mobile devices are needed (Korean Information Society Agency, 2011).

Mobile devices are key gadgets for communication in this modern era because of their high demand. Since mobile wireless devices are rapidly growing, communication experts should primarily consider increased mobile security measures. Mobile devices are confined to text messages and voice calls; additionally, they also store personal information, for example, a phone book, notepad and calendar. Presently, many organisations use these gadgets to store essential data. However, hacking of mobile devices and theft cases are reported due to their increased demand. Compared to wired devices, the security of mobile devices can easily be compromised due to several factors such as mobility and wireless transmission.

Hence, it cannot be overlooked that the device can be stolen or lost while storing data on it. Therefore, the security of the stored data must be ensured. The use of biometrics is a practical and technologically sound security measure for mobile devices. The security of mobile applications is enhanced due to this rapidly evolving technology. The enrolment and authentication are built into the architectural design of biometrics features that enable smartphone devices security. The correct matching of biometric data with the stored template is the responsibility of the authentication role. Hence, unauthorised access to the data will be prevented because there must be an actual match between the biometric data and the stored template.

In contrast, creating a template with the subsequent biometric information acquisition comes under the enrolment role. This practice will ensure that the information in the mobile device is only accessible to the administrator. Afterwards, a unique blend of characters is generated for every mobile device. However, the system administrator should ensure a clear password management policy. The data should be accessible to a person with valid credentials in this case. Furthermore, in the event of theft or loss of the gadget, the system administrator can remotely destroy the data.

1.2.1 Efficiency

Biometric authentication is not only secure and handy to carry out, but it is also efficient and intelligent. Biometrics based authentication systems use different unique attributes of persons, such as face, voice, and fingerprints. In addition, it is straightforward to upgrade the systems and speedily counteract the varying environments. Besides monitoring employee attendance, the network systems and information within an organisation are also secure when using biometric authentication. Compared with conventional authentication techniques, this method is more efficient due to the flexibility of the systems.

1.2.2 Compatibility

Different security requirements are associated with biometric authentication techniques. For example, some of the secured systems can also be accessed remotely. Therefore, biometric authentication and these systems are well-matched since an organisation finds it challenging to monitor all persons remotely accessing its system. Additionally, the systems also successfully correspond to several hardware and software platforms. As a result, it is easy to transform them, subject to security requirements.

1.2.3 Security Issues

Security issues linked with biometric authentication can be of two types, including concern with the assumption/model of biometrics and weakness of the biometric authentication system. Biometric systems can realise the security threats, such as identity theft or system compromise. A particular user has permanent biometric attributes. Therefore, if it is compromised, all the relevant applications would be affected by a biometric sample. Ultimately, the system or identity of the affected user would be influenced.

One of the most secure techniques for granting access to various systems is biometric authentication. Although it is safe, various security concerns are associated with it, i.e. biometric authentication systems hosting servers are also vulnerable to cryptographic attacks and viruses.

The security of a user or a system might be affected due to viruses and cryptographic attacks. Two or more forms of authentication should be integrated into a single system to improve its security.

1.3 Algorithms, Algorithmic Tools and Complexity

Algorithms are an integral part of computational theory. The Merriam-Webster dictionary defines an algorithm as solving a mathematical equation or a problem in a definite number of steps that usually includes a repeated function or process to conclude or solve a problem, especially by employing a computer (Merriam-Webster, 2016). In the algorithm, instructions are implemented in sequence for computations or solutions of a particular problem. These instructions are finite. The figure below demonstrates the example of an algorithm written for computing partitions by Guting, Behr, and Nidzwetzki (2021).

algorithm *SimilarityPartitioning*
input: S - a set of objects with a distance function d
  k - an integer parameter controlling the density of placing partition centers
output: PC - a set of partition centers
method:
  for each $s \in S$: let $r(s)$ be the distance of s to its k-th nearest neighbor within S;
  $PC := \emptyset$;
  for each $s \in S$:
    if $\forall p \in PC : d(s,p) \geq r(s) + r(p)$ **then** $PC := PC \cup \{s\}$;
  return PC
end *SimilarityPartitioning.*

Several inputs, instructions, statements, and functions can be seen in the above-demonstrated algorithm to generate the output in the form of computed similarity partitioning. As a result, an algorithm is a well-defined method for a computer to solve a problem. An algorithm is a set of actions that must be repeated in a particular sequence to solve a problem. A computer programme implements an algorithm, where accuracy, speed and memory are the determinants to evaluate each programme. Algorithms are not merely associated with computer science, but various scientific issues, e.g. security, biology, mathematical problems, design problems etc., can be

addressed through such a set of rules. Every computing device must follow some algorithm for performing its functions. Therefore, it can be said that algorithms form the backbone of the field. Algorithms aren't only for computer science; they may tackle a wide range of issues in science and other fields (e.g. security, biology ... etc.).

Several algorithms depend on their application areas; the most wellknown algorithms are Recursive Algorithms, Dynamic Programming Algorithms, Brute Force Algorithms, Greedy Algorithms, Divide and Conquer Algorithms, Backtracking Algorithms, and so on. Bioinformatics algorithms have been much researched together with their efficient computation applications, particularly the regularities in strings. It is recommended that security issues, such as biometrics be resolved by implementing efficient algorithms to compute cyclic regularities of strings. In Chapter 3, a fast minutiae extraction algorithm based on approximate circular string-matching was proposed that used the Needleman-Wunsch and Landau and Vishkin algorithms (Landau & Vishkin, 1998) was used to observe the alignment's accuracy by considering the most extended circular strings from the input and the stored fingerprint and asmf/acdmf (Advanced System Management Function/Advanced Collaborative Decision
Making Function) to check for any circular string rotation in the input, comparing it to the stored fingerprints (or vice versa).

A Maximal Motif Discovery in the Sliding Window (MMDSW) algorithm for real genomic data was proposed in Chapter 4. Fastening at the left end of both the text and the window is generally used to enable privacy difficulties with massive data, such as sliding window mechanism (Christian & Lecroq, 2004). Subsequently, the text in the window space and the pattern are compared to find matches or mismatches. After this attempt, the following text segment comes, where the matches or mismatches are re-checked. These processes will be repeated until all the text has been read. A suffix tree (ST) is utilised to determine the longest repeated suffix, which serve as a computation tool for solving motif discovery problem.

Bioinformatics is a molecular biology management information system with diverse practical applications (Luscombe, Greenbaum, & Gerstein, 2001), and it observes the processing of biologically derived information using specific algorithms or computer systems. The criteria to compare the algorithms are the performance assessment and their response regarding space requirement or processing time against different input sizes. The algorithm's efficiency can be evaluated by order of growth of its running time and facilitates comparison among the relative performances of corresponding algorithms. However, after ensuring the algorithm's correctness,

the algorithm's time complexity is usually determined. The 'programming tricks' have no considerable link with the development of efficient algorithms, but they operate well due to better data management (Bioinformatics Organization, 2016).

Therefore, structuring the information for efficient processing of data and its well-organised classification is referred to as the data structure.

Algorithm Analysis is a field of computer science that focuses on evaluating an algorithm's performance and comprehending its time complexity. In algorithm analysis:

'Big O' = The time complexity of an algorithm can be determined by counting the fundamental functions using basic operations.

T = Unit time is counted by the multiplication, subtraction or single addition.

F(n) = Function that determines an algorithm's complexity, where n equals the input size. By and large.

Algorithms are evaluated in terms of their performance, which includes response and processing time and the amount of working space required for various data sizes. The performance of an algorithm may be determined by the order in which the algorithm's running time grows. Hence, we can also have a performance comparison of similar algorithms. We can understand the algorithm through analysis, and knowledgeable enhancements can be recommended. Nonetheless, the correctness of the method for the given issue is verified to calculate the algorithm's time complexity.

Programme efficiency is one of the vital aspects of computer science. Programme efficiency is irrespective of the efficiency and memory size of processors, the robustness and the fast speed of computers, as these things do not reduce the magnitude of programme efficiency. Computer applications are rapidly growing. As a result, we perceive more data that is more complicated. Consequently, more advanced computing methods are necessary to store, analyse, and safeguard such massive data quantities. Therefore, the requirement for efficient algorithms is always in demand, which leads to the need for improved and highly efficient data structures. In addition, the data/information can quickly be processed and retrieved/extracted by selecting the appropriate data structure. To organise different kinds of data (images, videos, text, relational data, etc.), we need various data structures for different aims.

2 Preliminaries

This chapter provides some fundamental concepts and definitions following the introduction of bioinformatics and information security from an algorithmic view. Studying the cyclic factors of strings in this chapter led to the development of algorithms and implementing software applications in a cyclic pattern for biological data in Chapters 3 and 4.

2.1 Alphabets and Strings

Strings, which are data in the form of sequences or words, were employed in this thesis. The strings are ordered sequences of letters or symbols derived from a finite collection of characters or symbols known as the alphabet Σ. As a result, all files with a limited number of different characters may be termed strings.

Adenine (A), cytosine (C), guanine (G), and thymine (T) are four smaller chemical molecules (nucleotide bases) that make up the DNA sequence utilised in the thesis (As shown in Figure 2-1). The four letters A, C, G, and T are the DNA alphabet. An alphabet Σ is a nonempty finite set whose members are letters (or characters); the primary key of the alphabet set |Σ| indicates its size, and the symbol is used to denote it. Ordered alphabets (those with a full alphabet) and unordered alphabets (those without a complete alphabet) are both possible (usually known as general). An ordered alphabet is an integer alphabet representing the numbers 1 through σ. Furthermore, in this research, the symbol Σ will be ordered and of constant size (i.e., σ = O (1)). For DNA sequences, the alphabet is Σ = A, C, G, T, where σ = 4.

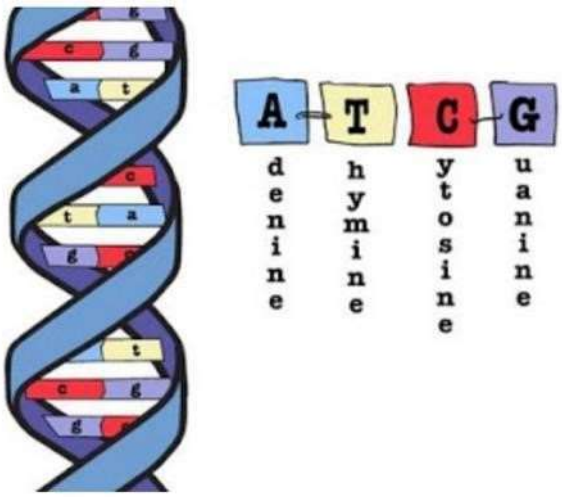

Figure 2-1 The four natural DNA letters

Stringology, or algorithms on strings, is a prominent topic in computer science that entails the study of strings. Zvi Galil created the word stringology in 1984 to describe knowledge of algorithms on sequences and strings. The core fields of stringology include bioinformatics, information security, and DNA processing, among others. This is projected to improve further due to the increased need for efficient high-speed stringology methods to solve difficulties such as searching for repeats in varied texts, approximate pattern matching, and so on (Fernandes, Pereira, & Freitas, 2009). Eventually, these issues and remedies will be discussed in Chapters 3 and 4 (Alshammary, Iliopoulos, & Khan, 2020; Alshammary, Iliopoulos, Mohamed, and Vayani, 2020).

2.2 Circular Strings

Circular patterns may be seen in genetics, such as fingerprint dermatoglyphics, viruses, bacterium, eukaryotic cells, and archaea DNA. Consequently, as described in, circular string algorithms seem to be crucial in the study of organisms with this structure. Circular strings have been investigated earlier in finding regularities in sequential genomic data. The job of discovering regularities in sequential genomic data may be simplified to the actual work of genome assembly and inference. Readers are directed to Fernandes, Pereira, and Freitas (2009), Thanbichler, Wang, and Shapiro (2005), Lipps (2008), Allers and Mevarech (2005), Gusfield, (1997a), Mosig et al. (2006), Lee et al. (2010) and reference to that for further information on the rationale and applications of this challenge in computational biology and other disciplines.

By linking P1 with Pm and establishing a cycle, the circular pattern C(P) matches to a given pattern P = P1...Pm is generated. As a result, we may observe the exact circular pattern as m various linear patterns, all regarded as comparable. Pattern matching among the text T and the circular pattern C(P) of a given pattern P focuses on the Circular Pattern Matching (CPM) issue. C(P) is seen as a collection of m patterns that begin at position j [1: m] and wrap around to the end.

To put it another way, in CPM, I look for all rotations of a specific pattern in a text. While CPM is a modification of the traditional pattern matching issue, there are other exciting variants. Because of the likelihood of data mistakes, the necessity of approximation or permitting inaccuracies in CPM is equally necessary and natural. Approximation permits a limit of errors within each occurrence rather than finding precise instances of all the rotations. This logical extension leads to the more fascinating and demanding approximate circular pattern matching (ACPM) issue.

2.2.1 Regularities

Usually, string regularity is associated with covers and periods that have been studied extensively, and broad algorithm applications are available, which allow better and more accurate computations of these (Defant, 2016; Erdős et al., 1973; Fujita, Magnant, & Ozeki 2010; Narayanan, 2017). Since the early 1900s (Thue, 1906). the research has been conducted upon a set of strings that have substrings (also referred to as periods or powers). Much significance is present for the different types of periods. Hence, they are considered numbers within computational biology and associated with various regulatory mechanisms essential for genomic finger printing (Kolpakov, Bana, & Kucherov, 2003).

For computation, various algorithms have proved to be effective for the strings' cyclic regularities computation. These computations evaluate cyclic periodicity, total cyclic periodicity, maximum local cyclic periodicity, cyclic covers. These calculations will be discussed in the current chapter. The results are detailed in the 15th International Conference on Artificial Intelligence Applications and Innovations (Ajala et al., 2019). For this work, the contribution made by the author was the strings, covers and periodicity cyclic regularities computations. They are stated briefly.

Definition 1: A string x of length |x|= n for a limited alphabet Σ, which consists of a set of characters, can be represented as x[1..n] = x[1]x[2]...x[i]...x[n], where $1 \leq i \leq n$ and the i-th letter of x is shown by x[i]∈ Σ.

Example 1: Consider a string *x* = aaabaabaabaabaaa is a non-empty string comprises Σ = {*a, b*} of length |*x*| = 16.

Definition 2: A substring of *x* [*i*·· *j*] is a suffix of x, if *j* = n, a proper suffix of x if *j* > 1 and a substring x [*i*·· *j*] is a prefix of x, if *i* = 1, a proper prefix of x if *i* < n.

***Example 2*:** If x = abbaaba is the longest proper prefix of x, then x [1, 5] = abbaa is the longest valid prefix of x. And the appropriate suffix of x is x [4, 7] = aaba.

Definition 3: For string x, a repeat would be the substring happening within x twice or more. It could be an overlapping, consecutive or discreet occurrence within string x.

***Example 3*:** When there is string *x = abcabcab*, the substrings *abc*, *abcabc* and the string *x would all be referred to as the* period of x; however, abc is the period of order *k* in string x, where *k* number of identical symbol blocks of a concatenation.

One of the most fundamental concepts in string regularities is period. For example, given a string x of length n, a period k of a string x is a sub string of x if it can be deconstructed into equal-length blocks of symbols, such that x= uku′, where u′ is a prefix of u. For simplicity, we'll ignore u′ and simply examine uk.

***Example 4*:** Consider a string x = aaabaabaabaabaaa = $u_1u_2u_3u_4$, where u_1 = aaab, u_2 = aaba, u_3 = abaa, u_4 = baaa and *k* = 4, ℓ = 4. So, x has a period of length ℓ.

Theorem 1: Given a string x of length n and an integer k, and $1 \leq k \leq n$, test whether k-cyclic periodicity, this can be determined in O (n/k) time and O (n) space.

Proof: We construct the suffix tree of x (see Crochemore, Hancart, & Lecroq, 2007; McCreight, 1976; Ukkonen, 1992; Weiner, 1973; Gusfield, 1997a). We let u = x [1 . . . k], then let ℓ_m represents the depth of the least common ancestor of x [1 . .n] (see Bender, 2000) , and x [i_m . .n]. We calculate the LCA ℓ_m of x [1 . .n] and x [i_m . .n] for i_m = 2k, 3k, . . . , and $\ell_k = n - k$, if $\ell_m = 1$ for some m, then x is not k-cyclic periodic string. Now take $C^{rignt} = (u\ell_m + 1 \ldots u_k)^R$, compute ℓ'_m the LCA of u R and C right m. If $\ell' m \geq \ell m$ for all m, then, x is k-cyclic periodic.

Theorem 2: When the string x of length n is given, the k-cyclic periodicity for all $1 \leq k \leq n$ could be tested in O(n log n) time and O(n) space.

Proof: We apply the algorithm of Theorem 1 for k = 1, 2, . . . , n and we test all cyclic periods of length k. The construction of the suffix tree of string x and x^R is done once costing O(n).
The total cost is

$$\mathcal{O}(n) + \mathcal{O}(\sum_{k=1}^{n} n/k) = \mathcal{O}(n \log n)$$

Lemma 1: Compute the cyclic period of x.

Proof: The smallest cyclic-period of x is the cyclic-period of x.

Definition 4: We define maximal local k-cyclic periodicity of a string x, if a substring y is cyclic periodic and y is not a substring of another cyclic periodic strings.

Example 5: Consider a string x=aaaababaaab, Σ={a,b}, and a substring y=aababababa is 3cyclic periodic and substring yα=aababababaa, α∈Σ, is not cyclic periodic, and substring βy=aaababababa, β∈Σ is not cyclic periodic. Therefore, the substring y=aababababa is maximal local 3-cyclic periodic in string x=aaaababababaa.

Theorem 3: We can compute all k-cyclic periodicity of x in O(nlogn) time.

Proof: We apply the algorithm for k=1,2,…,n and in this case, extend it to cyclic periods of length k+1, where |y| = m is cyclic periodic and yα = m+1 is not cyclic periodic. Next, we perform this algorithm on string x^R as $T(x^R)$, where |y| = m, again is cyclic periodic and βy = m+1 is not cyclic periodic. The construction of the suffix tree of string x is done once.
The total cost is

$$\mathcal{O}(\sum_{k=1}^{n} n/k) = \mathcal{O}(n \log n)$$

Lemma 2: Compute maximal local k-cyclic periodicity of x.

Proof: We compute and merge the arrays for yα and βy of x. That is the maximal local k-cyclic periodicity of x.

Definition 5: We say that a string x of length n is cyclic-coverable by a string u of length k′, if and only if, for every position i of x, the following condition holds x[β..γ]=c(u), 1≤β≤i≤γ≤n.

Example 6: Consider a string *x = ababbaba, then ab, abab, ababb, ababbab are covers of x.*

Example 7: Consider a string x = aababaa = c (u) = u_1u_2, u_1 = aaba, u_2 = abaa, $k' = 4$, $\gamma = 2$, is cyclic coverable by a string u, for every position i of x, x [1. .4] = x [4. .7] = c (u).

Definition 6: Compute all cyclic covers of a given string x, that is for all possible length cyclic covers.

Example 8: Consider a string x = ababbaba, then ab, abab, ababb, ababbab are covers of x.

Theorem 4: *Given a string* x *of length* n *and an integer* k′, $1 \leq k' \leq n$, *test whether it is* k′*-cyclic coverable, this can be determined in* O(n) *time and* O(n) *space.*

Proof: We compute the suffix tree of string x as T(x), and also we compute the suffix tree of string xR as T(xR).Then we check x[1,k′] with each one of x[n−k′+1,n], x[n−k′,n−1], x[n−k′−1,n−2] …x[2,k′+1], together with the reverse pairs in T(xR). This way we build a collection of cyclic covers if there is one. The construction of the suffix tree costs O(n); checking of equality costs O(1) and there are n factors. The total time is O(n).

Theorem 5: Given a string x of length n, test whether it is k′-cyclic coverable for $1 \leq k' \leq n$, this can be determined in O(n2) time and O(n) space.

Proof: We apply the algorithm for k′=1,2,…,n and we compare all cyclic coverable of length k′. The construction of the suffix tree of string x is done once. The total cost is

$$\mathcal{O}(\sum_{k'=1}^{n} n) = \mathcal{O}(n^2)$$

Lemma 3: Compute the cyclic coverability of x.

Proof: The smallest cyclic coverable of x is the all the cyclic coveralbe of x.

Definition 7: A seed is an extended cover in the sense of a cover of a super-string of *x*, A seed is a generalised cover: a substring *u* of *x will be considered a seed provided it acts as a cover whereby u shows an incomplete* first or last occurrence or both. An incomplete occurrence implies that the first occurrence is a non-empty *suffi*x and a last occurrence is a non-empty *prefix* of *u.* The substring *u* of *x is deemed to be cover where* concatenations and superpositions copies of *u result in the formation of x.*

***Example 9*:** Given a string *x = bbabab* has a proper seed *bab*, since *bab* covers a superstring *babbabab* of *x*.

Definition 8: A data structure comprising a collection of elements is known as an array. The data types of these elements are frequently the same, such as string or integer. Arrays are often used in computer applications to organise data and make finding and sorting a related group of variables easier. The array A containing a string x of length n has the following definition:

A[i]= ℓ, 1≤ i ≤ n, if and only if x[1..i] has cyclic periodicity ℓ by a string u and there no $u^{'}$, with $|u^{'}|\leq|u|$ that is a cyclic period of x[1..i].

Example 10: Given a string x = aababa of length 6, a cyclic periodic array A as follows:

x[1] = *a* =⇒ *A*[1]= 1 *x*[1..4] = *aaba* =⇒ *A*[4]= 1

x[1..2] = *a a* =⇒ *A*[2]= 2 *x*[1..5] = *aabab* =⇒ *A*[5]= 1

x[1..3] = *aab* =⇒ *A*[3]= 1 *x*[1..6] = *aab aba* =⇒ *A*[6]= 2

2.3 Circular String Pattern Matching for Fingerprint Recognition

Various concerns have been brought forward for several years regarding fingerprint recognition systems. Initially, it was quite simple, and two factors guided them. The first factor was the need to have, for example, a badge. This badge would be an essential identification tool that would allow the individual to enter a restricted facility. Another factor is that the individual knows the personal identification number to be used at the ATM or the password given to the user to ensure they have access to the restricted or confidential data.

Since technology evolves with time, it is necessary to have another third factor. This is representation through biometrics (Janker, 2002). With the help of a biometric security system, the citizen's privacy rights can be secured accurately and timely manner as the individuals are identified without requiring any paper or information such as a social security number (Zhang, 2002). With the help of this technology, users are empowered and can prove to others who they are by using their biometrics. In this case, the users can prove themselves but not reveal the biometric information. If there is a positive match, then the user's record would be encrypted and displayed. If there is no positive match, then the user data would remain inaccessible to the system administrator (Li & Jain, 2009). Hence, the overall identification procedure has been strengthened.

Figure 2-2 An example of types of biometrics

In Chapter 3, To solved the challenges associated with fingerprint identification circular string-matching utilising approximate circular string-matching algorithms, I have implemented Fast Minutiae Extraction algorithm (Alshammary, Iliopoulos, & Khan, 2020) this would assist alleviate the problem of rotation in a fingerprint identification system by matching the fingerprint data using approximate circular string-matching. The Needleman Wunsch and Landau and Vishkin algorithms were used to ensure that the alignment is precise, and the asmf/acdmf (Advanced System Management Function/Advanced Collaborative Decision Making Function) to ensure that the circular string relative rotation input matched with the saved fingerprint image to extract all rotation occurrences.

Definition 9: To see whether a pattern p appears anywhere in text t, find the maximum number of k-differences between the pattern p and text t (Clifford & Iliopoulos, 2004). Find all factors F of T such that $F \equiv k\ P^i$ for any $0 \leq i < m$, and when I have a factor F = T [j: j + |F| 1] such that F k Pi let's state that the circular pattern C(P) k-matches T at position j. Given a pattern P of length m, a text T of length n > m, and an integer threshold k < m, this match is also due to Pi, the ith rotation of P.

Example 11: If p = bab and t = aababxbababx, p may be found within t starting at locations 3, 7, and 9. It's worth noting that two instances of p might potentially overlap, as shown at places 7 and 9 (Zhang & Yan, 2001).

Definition 10: A string *x* of length |x|= n, and cyclic factor u of length k, we denote by c(u)= $u_1u_2...u_n$= x, $1 \leq k \leq n$, the *i*-th rotation of x and c (u) =x.

Example 12: Consider a cyclic factor u of length k, where u= ababc and c(u) is one of the rotations u_1=ababc, u_2=babca, u_3=abcab, u_4=bcaba, and u_5=cabab.

The definitions above, consider the problem of finding occurrences of a pattern p of length m with circular structure in a text T of length n (Barton, Iliopoulos, & Pissis, 2014), in which n is the concatenation of every string representation of the database fingerprints and m is the string representation of the fingerprint to be identified, based on the definitions above. The complexity of this approach is O(n).

2.4 Circular String Pattern Discovery for Biological Data

Viruses with circular strings served as motivation for the string cyclic factor. For instance, the DNA sequence of Escherichia coli (E.coli) has 154 circular bases (Wurpel, Beatson, Totsika, Petty, & Schembri, 2013) (Figure 2-3).

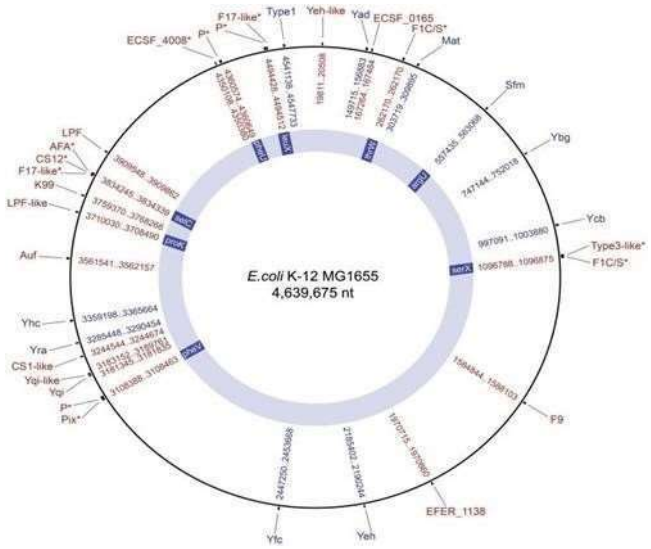

Figure 2-3: The E. coli K-12 MG1655 chromosome

The E. coli K-12 MG1655 chromosome (outer black ring) was used as a reference map to visualize the locus position of 30 chromosome-borne CU fimbrial types. Types highlighted in blue are present in E. coli K-12 MG1655, types in red are absent in this strain. Fimbrial types associated with PAIs are indicated by an asterisk. A number of PAI associated fimbrial gene clusters occupy different locus positions relative to the MG1655 genome. tRNA sites that flank CU-containing PAIs are indicated on the inner blue ring (Wurpel, Beatson, Totsika, Petty, & Schembri, 2013).

The viruses break up at any circle's point, for instance, that can appear in the DNA sequence as xδ…xnx1…xδ−1 breaking up at position δ (Figure 2-4).

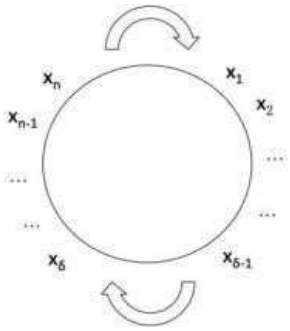

Figure 2-4 Circular pattern

2.4.1 DNA, RNA, and Protein Sequences

The DNA structure was found in 1953 by James Watson, Francis Crick, Maurice Wilkins, and Rosalind Franklin. Many DNA molecules consist of a double-stranded helix that includes two long biopolymers with nucleotides.

Definition 11: A DNA molecule is composed of four subunits (nucleotide bases): adenine (A), cytosine (C), guanine (G), and thymine (T); the four letters A, C, G, and T are referred to as the DNA alphabets. Nonetheless, each of the four nucleotide bases forms a base pair when connecting with its counterpart base. As a result, A and T make a bond; similarly, G and C form a bond.

Example 13: The DNA sequence ATGATTGACAT has been provided, indicating that TACTAACTGTA will be the complementary strand sequence. Hence, the base sequence is used by the DNA to store data.

The DNA sequence contains four nucleotides ordered in the DNA molecule bases. Genetic information is present within the DNA segments and is called genes. Within a single DNA, there is a string with four letters. If one DNA strand sequence is identified, it is possible to predict the strand sequence paired or complemented with.

The nucleotide bases reading and establishing the nucleotide bases' order in the DNA molecule is DNA sequencing. In 1975, the sequencing procedure was initially developed by Frederick Sanger (Sanger Sequencing).

This was a laboratory procedure (Nature Education, 2016) and quite time consuming and expensive (Prober, 1987). For DNA sequencing, the technology used was Sanger Sequencing which is why it took several years for it to be replaced by a new DNA sequencing technology that was brought forward during the 1990s (Antonio, 2009). The genomic sequences' production volume increased, and it was now applicable to genomics within medicine, forensics, evolutionary and molecular biology and other fields. An associated influence was that over the past 15 years, there had been an explosive rate of sequences being added to databases like GenBank (GenBank and W. Statistics, 2020). The growth of genomic data sequences within GenBank and WGS databases can be observed in Figures 2-5.

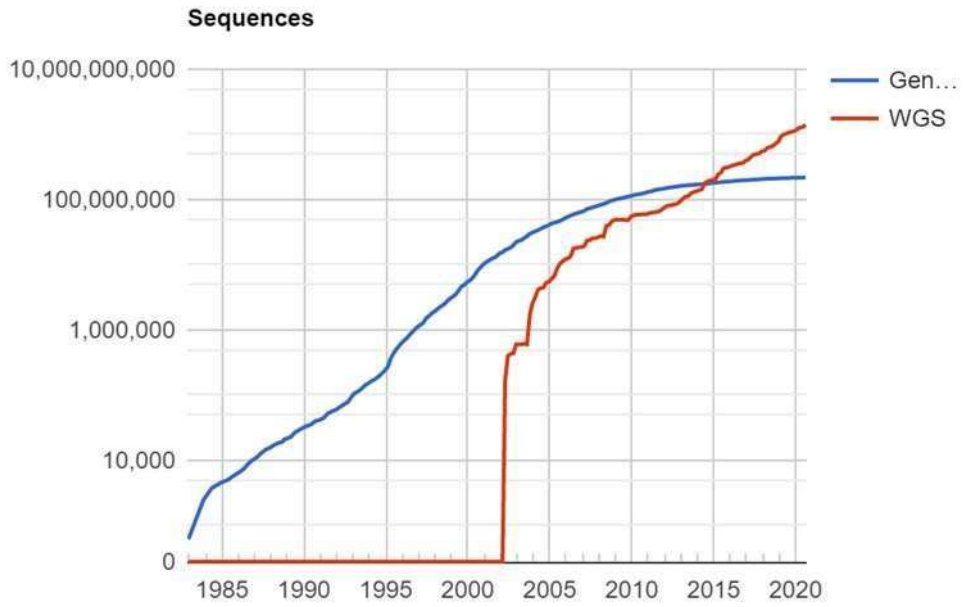

Figure 2-5 Genomic data growth in GenBank and WGS databases. Data retrieved from GenBank

For sequencing procedures, the essential variables are speed, cost, the quantity of data, error rate, and sequence length (Pissis, 2016).

The earlier sequencing features and genome sequencing have been revamped using new throughput sequencing technology procedures. This is accomplished by analysing tens of millions of small sequences (reads) included inside a single experiment. As compared to the earlier methods, this turns out to be cheaper. Since the data generated are extensive, there is significant demand for an appropriate algorithm for mapping these short sequences to the reference genome (Antoniou et al., 2009). Section 1.3 indicates that the algorithm uses bioinformatics for specific computers to process biologically extracted information like DNA sequencing.

Much significance has been granted to DNA. The gene sequence behaves as a language that directs cells to create specific proteins.

The signals from the genes may be translated into the amino acid sequence of the protein using the intermediate language stored in the Ribonucleic Acid (RNA) sequence.

The protein determines the characteristic, referred to as life's central dogma (As shown in Figure 2-6).

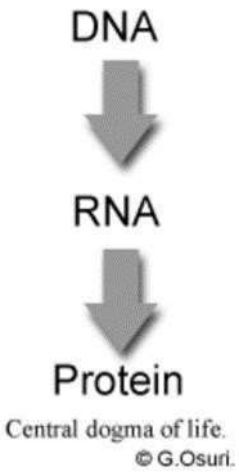

Figure 2-6 Central dogma of life (Bioinformatics Organization, 2016)

Notes: For the production of the particular proteins, the cells are instructed by the DNA sequences genes. This would then establish traits. The strings of chromosomes are genes. The alterations in the genes' DNA sequence are mutations.

DNA and RNA have a lot in common. Both have a sugar-phosphate backbone and are nucleic acids with nitrogen-containing bases. The structural and functional distinctions between DNA and RNA are used to distinguish them. The single-stranded nature of RNA contrasts with the double-stranded nature of DNA. Thymine is found in DNA, whereas Uracil is found in RNA. The nucleotides in RNA have sugar ribose, and DNA has Deoxyribose. Keeping the functional aspects in mind, the protein-encoding data is present in DNA, and RNA uses the information to establish a cell for the production of the particular protein.

There are differences between DNA and RNA that have been discovered. The fact that RNA is single-stranded and DNA is double-stranded is the most essential difference. In addition, DNA contains deoxyribose sugar, whereas RNA has ribose sugar. These two sugars are necessary components of nucleotides. There is also a difference in the bases. DNA has the thymine nucleotide, whereas RNA contains the uracil base. Transcription of RNA is due to the enzymes in DNA referred to as RNA polymerases, and it is additionally processed through other enzymes. The process each molecule goes through is another DNA and RNA difference. Replication is present in DNA, and translation occurs in RNA. The biological procedure of DNA replication through DNA identical replicas forms a single original DNA molecule.

In RNA, a translation process occurs where the mRNA is decoded to establish a protein that attains a specific amino acid series. Lastly, within the nucleus, there are various processes that the DNA goes through. For RNA, the processes take place in the cytoplasm and nucleus.

RNA	DNA
Single-Stranded	Double-Stranded
Has Uracil as a base	Has Thymine as a base
Ribose as the sugar	Deoxyribose as the sugar
Uses protein-encoding information	Maintains protein-encoding information

Figure 2-7 Differences between DNA and RNA (Bioinformatics Organization, 2016)

Notes: Genetic data is stored in the DNA, but RNA uses the data to assist the cells in creating proteins.

Definition 12: DNA and RNA maintain a similar structure. A single base, Uracil (U), creates the alphabet difference and T is replaced. Hence, the alphabets of RNA are A, G, C, and U, in which A and U are bonding.

Example 14: The Watson and Crick model suggest that the template strand carries the gene sequence The coding strand produces a T C G G G T A. Complementary pairs with the template strand.

T A G C G C A T is the base sequence of the strand. The transcription procedure is facilitated by binding the RNAP with the DNA sequence promoter region. When the promoter site and RNAP are attached, the primary transcript is formed by transcription through the template strand, which maintains the base sequence as U A G C G C A U. (As shown in Figure 2-8)

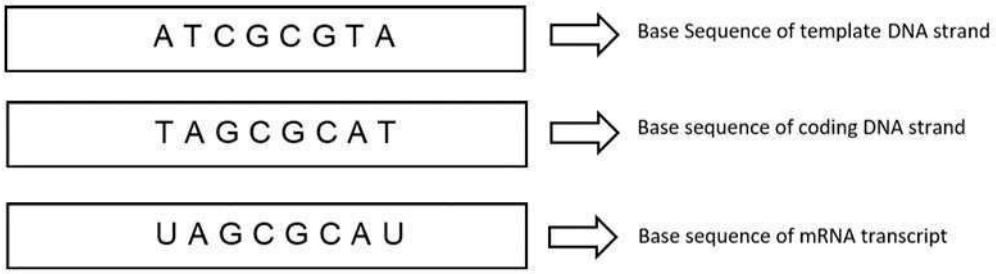

Figure 2-8 Watson and Crick model of DNA (Watson & Crick, 1953)

Proteins are biomolecules that have a variety of roles in the body. A protein is built up of a series of smaller components called amino acids that are folded together to create complicated three-dimensional structures. Up to 20 different amino acids are used to make the bulk of proteins. As a result, a protein molecule may be thought of as a string of 20 letters across an alphabet. Protein sequences are encoded in genes, which are DNA sub-sequences. Exons are small substrings of DNA interspersed with introns, which are longer substrings. A transcript is a logical grouping of exons that represents a single protein. Numerous transcripts from the same gene may code for multiple proteins. *Transcription*, splicing, and *translation* are the three phases of DNA translation into protein. During *transcription*, the two strands open up and form a complementary (in one of the strands) RNA molecule. The only difference between RNA and DNA is that T is replaced with another nucleotide, U. (i.e. Uracil). The introns are then spliced (i.e. cut off) to combine subsets of exons to form one or more transcripts. mRNA is produced due to the splicing process (matured messenger RNA). Translation is the last step, in which the mRNA is sequentially read from left to right, and a triplet of bases (known as a codon) is

transcribed from left to right into exact amino acids. The appropriate protein is then created by chaining them together. The genetic code, which links such triplets to amino acids in most animals, is known as the genetic code.

The binding of particular proteins, known as transcription factors, in relevant regions known as regulatory regions, regulate the transcription rate (by inhibiting or increasing it). Transcription factors bind to DNA substrings, referred to as transcription factor binding sites (TFBS). These may be found in the promoter region (a 100-1000 base-pair length area that starts the transcription process and is near the gene transcription start point) or at a substantial sequential distance from the gene.

2.5 Strings Similarity

Strings similarity is a bioinformatics' term that determines the similarity in the alignment of two strings. It may also be used on a single string to determine the parts of the string that are similar, the extent to which the two strings are similar or the distance between them. This can be done in the following manner.

2.5.1 Sequence Alignment and Dynamic Programming

The alignment of DNA/RNA or protein sequences to detect regions of similarity is described in bioinformatics. It's utilised to determine how the sequences are related functionally, structurally, and evolutionarily. Alignment determines how similar the query sequence is to the different database sequences. The basis of the Needleman-Wunsch and Landau and Vishkin algorithms (Landau & Vishkin, 1988) is the dynamic programming method, in which the problem is disintegrated into smaller independent sub-problems. Scores are allocated to determine more quantitative alignment. In the Needleman-Wunsch Algorithm, aligned sequences are usually presented as rows in a matrix of size O (nm). Gaps are added between the residues to align similar characters in successive columns. A gap score refers to the penalty awarded to alignment following insertion or deletion. Three steps are part of the dynamic programming matrix:

1. The initialisation of the matrix with the potential scores.
2. Filling of the matrix with the highest scores.
3. Tracing back of the residues for suitable alignment.

Arrays in programming store several types of data that are of the same kind. For example, it helps the programmer store several elements that are of the same type. All the elements are stored by the array, rather than considering all the same type. It is challenging and time-consuming to store a thousand integers or strings; therefore, it is simpler to work with arrays. Array List class can be resized and is found in Java. Array List Class is dynamic, which means that elements can be inserted or eliminated whenever required. Arrays are used by all programming languages like C sharp, Java, C++ and Python. In general, an array is a container that includes several sections, and each section signifies an element.

Developers should be aware of the number of elements that can be held by an array when developing an array. In addition, the data type of the element should be specified by the programmer. A single variable number is used to represent arrays, and the number of elements should be added in braces for example:

$A=\{1,2,3,4,5,6\}$

The size of the array should be an integer that has a value that is higher than zero. Index array names access elements of an array. Apart from indexes, data stored in an array are also accessed by the subscript. Contagious memory locations are part of all arrays. The Last element is represented by the highest address, whereas the lowest address signifies the first element. There are two kinds of arrays: one-dimensional and multi-dimensional arrays. Loops are used to demonstrate array elements. Pointers are used to manipulate arrays and are also transferred to a function to be accessed in a function. On the whole, arrays are employed in all programming languages to consolidate several data objects into a single data structure.

The purpose of creating anonymous arrays is to use them immediately, and they can be named later by the programmer based on the requirement. In Java, the arrays are transferred to a method to become efficient enough to be reused. The array class is a Java package that includes overloaded techniques to generate and manipulate arrays. The class name accesses static techniques of the class array. Array manipulation techniques like sorting, searching and copying are offered by Java SE.

Array List Class is a collection framework that is available in Java and has a dynamic characteristic. In contrast to standard arrays, when the size increases, the array class list increases automatically. When objects are eliminated, the arrays decrease. An array list is preferred by the majority of software engineers instead of arrays because, in programmes that have extensive manipulation, it is pretty efficient. The list interface is used by the array list, including several techniques developers use to execute operations. On the whole, an array list is extensively used by programmers due to its flexibility and functionality.

Arrays work well for memory allocations and may be employed at any place in the code, because of which they are appropriate for several programming languages. Arrays are also the most critical data structures that are used for adjusting and accessing data objects.

2.6 Searching and Sorting Algorithms

The sorting algorithm is one of the essential functions of computer science. Most programmers use sorting as a pre-step before beginning to solve any computational issue to improve the algorithm's execution time. The radix sort, one of the quickest linear time sorting algorithms, is an example of a sorting algorithm.

The items are sorted by the majority of the algorithms, particularly the searching algorithms, before carrying out any other instructions. For instance, consider the binary search that is considered one of the fasting algorithms for performing a search in a sorted array (Skiena, 2008).

2.6.1 Application Testing and Scoring

Application testing and scoring assess the security of applications and make sure that the application complies with the security requirements formulated by the public safety organizations. The evaluation process may adopt standard criteria for evaluating applications and measuring their performance to the security controls. This process may comprise risk analysis, describing the possible risks of application. The evaluation outcome should be a score that represents the sensitivity of the application and the degree of impact related to such risks. The applications that successfully move past the evaluation would be allowed a protection profile level (score). The scoring grid can be considered a suitable metric that can allow public safety organizations to comprehend the type of risks involved in the application, the risk impact level, and the extent to which the application is secure. Therefore, public safety organizations

should have faith in the evaluation and scoring process and use it to determine whether they should accept or reject the application. The score may then be used as an input to facilitate the security assessment and authorization (Communications Security Establishment, 2020) process for national security systems that can benefit from previously tested applications. Therefore, it would be important to develop the application score to facilitate the testing applications while the vetting process is being carried out.

A series of security controls may be used as a guideline during the assessment procedure to identify whether the applications fulfil the security controls and the specified security requirements. Security controls signify the 'safeguards/countermeasures recommended for organizations to assess applications in terms of the protection level of the integrity, confidentiality and availability of information stored, processed and transferred by those applications'. Therefore, the organisations can determine the assurance level fulfilled by the applications after implementing the security controls. In the same way, a protection profile level (score) is allocated to every application that shows their adherence to such security measures. This score may then be used as part of the national security systems' security assessment and authorization (SA&A) procedure.

The application assessment process commences with the applications being uploaded by the Application Developer, after which the analyzers are to test and give it a score, which may involve a tool, services and human elements. After receiving the application, a secured database or respiratory should be used to store the application to ensure that there is no unauthorized access to the application which may give rise to possible integrity issues (such as changes to the application), or infringements of intellectual property rights by gaining access to the source code or decompiled code of application.

Once the application is processed by the analyzer, it will be tested for threats to the software that may lead to violations of certain security controls and requirements. If the analyzer is a third party organization or an individual external to the public safety organizations, then it is the responsibility of the analyzer to make sure of the confidentiality, security and integrity of the application files with respect to sharing, storing and processing, while making sure that they conform to the specified agreements. In this manner, public safety organizations can have faith in the evaluation and scoring process, which is believed to facilitate organizations in taking decisions regarding whether the applications should be accepted or rejected.

The analyzer then carries out a report and risk assessment to facilitate national security systems' security assessment and authorization (SA&A) process. The software vulnerabilities and risks are identified in the report generated from this assessment. A score is also calculated from the risk assessment, which shows the probability that the software threats and risks identified may be exploited and how this would affect other applications, devices, and information. The report supports the administrators or security personnel and risk assessment in validating the specific security controls and requirements and help approvers in determining whether an application should be accepted or rejected.

To determine the extent to which public safety mobile applications conform to public safety security requirements, they need to undergo an extensive application testing process. Furthermore, there should be a process for rating applications within the application store security (such as star rating) to enable public safety organizations to comprehend the application's security level and correspondingly enable or prevent the members from installing applications from installing applications the application store.

ISST put forward recommendations regarding the testing of mobile applications, including the guidelines pertinent to the testing of applications, testing methods that facilitate the SA&A process, and assessment tools and methods employed while performing application testing. Rigorous testing of the application is carried out during the testing process to identify software vulnerabilities and security threats to determine that the public safety security requirements are fulfilled by the application. Through the testing process, the integrity of the applications being tested will be confirmed with respect to protection of the information accessed, permission required and information stored. A 'Mobile Security Project' was put forward by the Open Web Application Security Project (OWASP) the purpose of which was to offer standards to the organizations pertaining to mobile application testing and to standardize the testing mechanisms of applications that may be customized to fulfill the security requirements of every organization. The objective of the project is to offer guidelines for Application Developers as well as organizations' security testers. OWASP asserted that the guidelines offer 'a mobile evaluation that integrates dynamic analysis, forensic analysis and static analysis to make sure that protection is provided to most of the mobile application attack surfaces. Mobile source code, disassembled or decompiled code testing, is included in the static analysis. The examination of the application's local inter-process, remove service dependencies and forensic assessment of the local file system are part of the dynamic analysis.

2.6.2 Applications Information Sharing

Access and control management is offered by access management methods, which makes it essential for each mobile application to obtain a certification or license so that they can gain access to information.

Apprehensions are raised by mobile devices pertaining to the integrity and security of information, with respect to the risks, vulnerabilities or possibility of compromise, or monitoring of devices because of the nature of mobility and complexity of attacks. Furthermore, a few organizations may permit Bring Your Own Device, a decision that makes it more difficult to offer a secure environment. Existing mobile devices are usually not able to offer security guarantees to users and organizations because of their inadequate security capabilities. The existing mobile security techniques are not sufficient to decrease the associated threats and risks. In addition, serious security risks can be caused by mobile applications as they may include vulnerabilities that intruders can use to gain unauthorized access to device resources, which includes sensitive information stored in the mobile device. In addition, it may make the information infrastructure vulnerable to possible security attacks.

There continues to be an increasing dependence on mobile devices and applications. This gives rise to a greater number of security threats, because of which the demand for a more secure mobile application framework in government and the public safety community has increased. Each of the security and interoperability requirements affects the need for best practices, standards, security technologies and controls that are suitable for the requirements of organizations, and which should be formulated and adopted to regulate mobile device security and mobile application security that is according to the government and public safety environments.

The requirements of public safety organizations with respect to interoperability and security can be managed by examining the existing security controls, technologies and practices that are modified to enhance the security of mobile devices and applications.

Furthermore, the efforts being made presently for adopting a mobility ecosystem to fulfil the security needed for mobile devices and applications implemented by the government and public safety organizations were also examined.

2.7 Fundamental Data Structures

2.7.1 Suffix Trees

The general idea about the suffix trees was extracted from Crochemore, Hancart, and Lecroq (2007), McCreight (1976), Ukkonen (1992), Weiner, (1973), and Gusfield (1997a). This thesis uses suffix trees as computation tools.

The Suffix Tree**,** also called PAT Tree, contains all the text suffixes according to their values and positions. It is a compressed trie. The primary motivation behind the suffix tree is the implementation of critical string operations quickly. Properties of Suffix tree include:

Suffix Tree for a text X
Text Size n
From an Alphabet of Size d
Stores all the $\frac{n(n-1)}{2}$ suffixes of X in $O(n)$ space
Can be constructed in $O(dn)$ time
Supports artibitrary pattern matching and prefix matching queries in $O(dm)$ time
Where m is the length of pattern

Trivial algorithm to build suffix tree is:

1. Generate all suffixes of given text
2. Consider all suffixes as individual word
3.Build a compressed trie

Example 15:

Consider the example of word “banana\$”
where
$ is string termination character
suffixes of "banana\$" are
banana\$
anana\$
nana\$
ana\$
na\$
a\$
\$

For pattern search in above-illustrated suffix tree:

Path: $ No match found, move back
Path: a No match found, move back
Path: banana$ No match found, move back
Path: na partial match found, search deeper
Path: na$ No match found, move back
Path: nana pattern found

The most important piece of information associated with Suffix tree is Longest Common Prefix (LCP) array. LCP is an array where each index stores how many characters, two sorted suffixes have in common to each other. Following example best shows the concept of the LCP array; Let us find the LCP array of string ABABBAB.

Table 2-1: LCP array

Sorted Index	*LCP Value*	*Suffix*
5	0	*AB*
0	2	*ABABAB*
2	2	*ABBAB*
6	0	*B*
4	1	*BAB*
1	3	*BABBAB*
3	1	*BBAB*

Hence, in the LCP array, every index tracks the common characters in two adjacent and sorted suffixes. In $0(n\log(n))$ and $0(n)$ there are many ways to construct the LCP arrays. It is fine to set LCP array as *LCP*[0] for many purposes but by convention *LCP*[0] is undefined.

Pattern searching, seeking the longest repeated substring, lowest common ancestors, accurate string matching, and searching the longest common substring are some of the applications of suffix trees.

Every internal node, save the root, has at least two children, and every edge is given the name of a non-empty factor of x. Any two edges from a node cannot have edge labels that begin with the same letter. If v is the node of ST (x), then v's path-label is the concatenation of the edge labels along the route from the root to v: the length of the path labels is the node v's string-depth. In the case of any i, $1 \leq i \leq n$, is precisely the suffix x [i..n]. Remember that each terminal is a leaf if the last letter of x is different, i.e. each suffix ends in a lead node.

To achieve a one-to-one correspondence between the suffixes and leaf nodes, a unique symbol (typically a '$' such that $\$ \in \Sigma$) is added to x. (see Figure 2-9).

Figure 2-9 A unique symbol to create suffixes from sequences

Farach (1997) gave the first suffix tree construction algorithm that is optimal for all alphabets. In particular, this is the first linear-time algorithm for strings drawn from an alphabet of integers in a polynomial range. Farach's algorithm has become the basis for new algorithms for constructing both suffix trees and suffix arrays, for example, in external memory, compressed, succinct, etc.

In older linear-time construction algorithms, suffix-links were an integral attribute; however, recent algorithms are allocated based on suffix links formulated on Farach's algorithm (Farach, 1997). Every internal non-root node has a suffix-link to another internal node in a full suffix tree. If the string Xα is represented in the route from the root to a node, it consists of a suffixlink to the internal node marked α, where X is a single character and α is a string (probably an empty string).

The suffix-link from the node for ANA to the node for NA is shown in the (example 15) above. Similarly, suffix-links are used in a few algorithms that operate on the tree.

Functionality

- A suffix tree for a string S of length n may be constructed in O (n) time if the characters are obtained from an alphabet of integers in a polynomial range (particularly for constant-sized alphabets). The majority of the time spent running larger alphabets is spent categorising the letters to incorporate them into a range of size O (n), which requires O (n log n) time.
- In the scenario where W is the LCP of u and v for any two suffixes and For u and v, the route in S (x) corresponding to W is similar. This signifies that the string depth of the two leaves' Lowest Common Ancestor (LCA) node is comparable to the suffixes' length of LCP denoted by those leaves.

2.7.2 Suffix Links

Linking each internal node of an implied suffix tree are pointers that are known as suffix links. They allow a lesser time complexity O (n), space and linear time. This is beneficial in the sense that the increase in speed will lead to the required linear-time algorithm.

***Example 16*:** Consider a Suffix Tree (X) = ABABABC; the dotted lines indicate the suffix links.

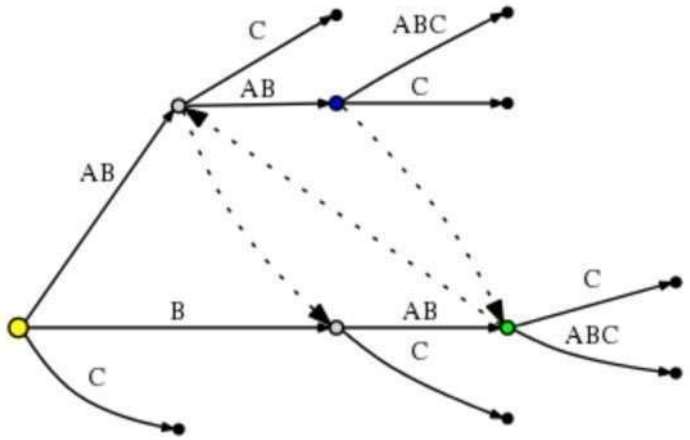

Figure 2-10 An example for suffix links

Suffix connections save time by traversing the tree quicker from the root to the suffixes of S. Using one of the numerous seminal methods, the suffix tree S(x) creation requires O (n) time and space. To swiftly carry out string operations, suffix trees belong to a joint group of data structures that store a set of strings. Among other data structures, Patricias are pretty notable as they establish Patricia trees as a pointer for looking within the marked-up text. Other data structures like suffix trees are:

- Trees
- Keywords trees
- Radix trees, the adequacy of suffix trees' application resides in exhibiting the internal structure. In string pattern recognition, this portrayal of the string structure effectively leads to problem-solving. We will explore the following points in Chapter 4: The substring problem for a database of patterns
- Computation of palindromes and the DNA contamination problem
- Solution of the Maximal Motif Discovery for Biological Data in a Sliding Window problem (MMDSW)
- The longest common substring of two strings

Suffix tree is a compressed trie of all the suffixes of a given string. Suffix trees help in solving a lot of string related problems like pattern matching (in chapter 3), finding distinct substrings in a given string (chapter 4), etc. Suffix tree often takes place in bioinformatics. Suppose that the text is a substantial database where there are several DNA sequences, assembled by read multiple fragments of sequences and then link them back together in the correct order.

3 Fingerprints Recognition System-Based on Mobile Device Identification Using Circular String Pattern Matching Techniques

This research study was published and presented at the 16th International Conference on Artificial Intelligence Applications and Innovations (Alshammary, Iliopoulos, & Khan, 2020). The author's personal contribution to the work was in refinement to the design of the algorithm, as well as its implementation and experimentation.

3.1 Introduction

The advancement in technology has made the sharing and transmission of data across mobile devices more transparent and accessible to unknown organizations. This increase in data transmission has influenced manufacturers to develop more accurate recognition systems.

Automatic person identification systems based on biometrics are more reliable since the use of different unique attributes of persons, such as face, voice, and fingerprints. In addition, it is straightforward to upgrade the systems and speedily counteract the varying environments within an organisation are also secure when using biometric authentication compared with conventional authentication techniques, As biometric authentication is more efficient and intelligent due to the flexibility of the systems, it is not only secure and handy to carry out (Sebastian, 2013; Unara, Senga, & Abbasi, 2014).

The use of biometrics is a practical and technologically sound security measure for mobile devices, where the security of mobile applications is enhanced due to this rapidly evolving technology, by enhancing the security issues associated with biometric authentication, which are of two types:

- Concern with the assumption and model of biometrics system.
- Weakness of the biometric authentication system.

Biometric systems can realise the security threats, such as identity theft or system compromise. Therefore, if it is compromised, all the relevant applications would be affected by a biometric sample. To avoid this, the enrolment and authentication are built into the architectural design of biometrics features that enable smartphone devices security, through the correct matching of biometric data with the stored template is the responsibility of the authentication role.

Hence, unauthorised access to the data will be prevented because there must be an actual match between the biometric data and the stored template.

There are various types of biometrics available in which finger-based identification is the most used and reliable one, Fingerprint recognition systems are divided into two categories:

- Verification
- Identification

In verification, the ID (identity) corresponds with the input fingerprint which is then confirmed by the system to either correspond with the saved fingerprint of that ID or not with a positive or a negative response. On the other hand, in identification, the system generates a list of fingerprints against the input and identifies the ID. In that case, the response is short and, in most cases, a list of empty fingerprints.

Instead of focusing on the rotation of fingerprints, the current studies assume that the orientation of the printed finger will align with the orientation of the target fingerprint image; This reduces the accuracy of fingerprint recognition across a variety of fingerprint scenarios and orientations. As this area of research has become imperative and needs focus as modern mobile devices and computers have adopted fingerprint recognition as a way of authenticating individuals (Alatabbi, 2014). Most recently in (Ajala, Iliopoulos, & Khan, 2016), the authors have presented an algorithm for approximate circular string matching, with an average-case runtime of O(n) time, where m is the length of text x. In this chapter I propose and present an algorithm (Alshammary, Iliopoulos, & Khan, 2020) with better experimental run time for searching patterns in circular string, which runs in linear time, O(n).

3.1.1 Chapter Summary

In this chapter I present a fast minutiae extraction algorithm for the approximate circular string pattern matching problem of fingerprints recognition system- based on mobile device identification. In particular, I employ the minutiae extraction of circular strings to identify the fingerprint with the relative orientation. I have done experimental studies to compare my algorithm with the state-of-the-art algorithms and the result of my algorithm turns out to be much faster (As shown in Figure 3-11) in practice because of the huge reduction in the search space. I propose and present an algorithm (Alshammary, Iliopoulos, & Khan, 2020) with better experimental run time for searching patterns in circular string, which runs in linear time, O(n).

The rest of the chapter is organised as follows: Section 3.2 providing the background and reviewing related literature. Section 3.3 gives a problem definition and preliminary description along with the terminologies and concepts related to stringology that will be used in this chapter. Section 3.4 presents the contribution. Section 3.5 presents the experimental results. A brief conclusion in Section 3.6.

3.2 Background and Related Work

Fingerprinting has been used since the 19th century. It is a way of identifying a person and has been utilized for a long time. The applied capacity of this method is alluring for overseeing and keeping a track on criminal records. Thanks to this method the efficiency of the local police increased, and it helped progress forensic medicine, which is evidenced in recorded criminal cases in Argentina (Nature Education, 2016; Unar, Seng, & Abbasi, 2014). This technique is now used in many organizations including law enforcement, industries and for commercial applications, like for financial transactions, access control and even in mobile devices and computers.

Fingerprinting is classified by physiological attributes, there are three major kinds of features of a fingerprint pattern (FP), which are:

- Whorl
- Loop
- Arch

Figure 3-1 Three major kinds of features for fingerprint patterns (FP)

Henry's classification (Henry, 1900) is known to be the most established and known way of characterizing fingerprints, The AFIS (Automated Fingerprint Identification System) classification methods were based on Henry's classification (Henry, 1900). There are eight classes of fingerprints (As shown in Figure 3-2):

- Plain Arch
- Tented Arch
- Ulnar Loop
- Radial Loop
- Double-Loop Whorl
- Plain Whorl
- Central-Pocket Whorl
- Accidental Whorl

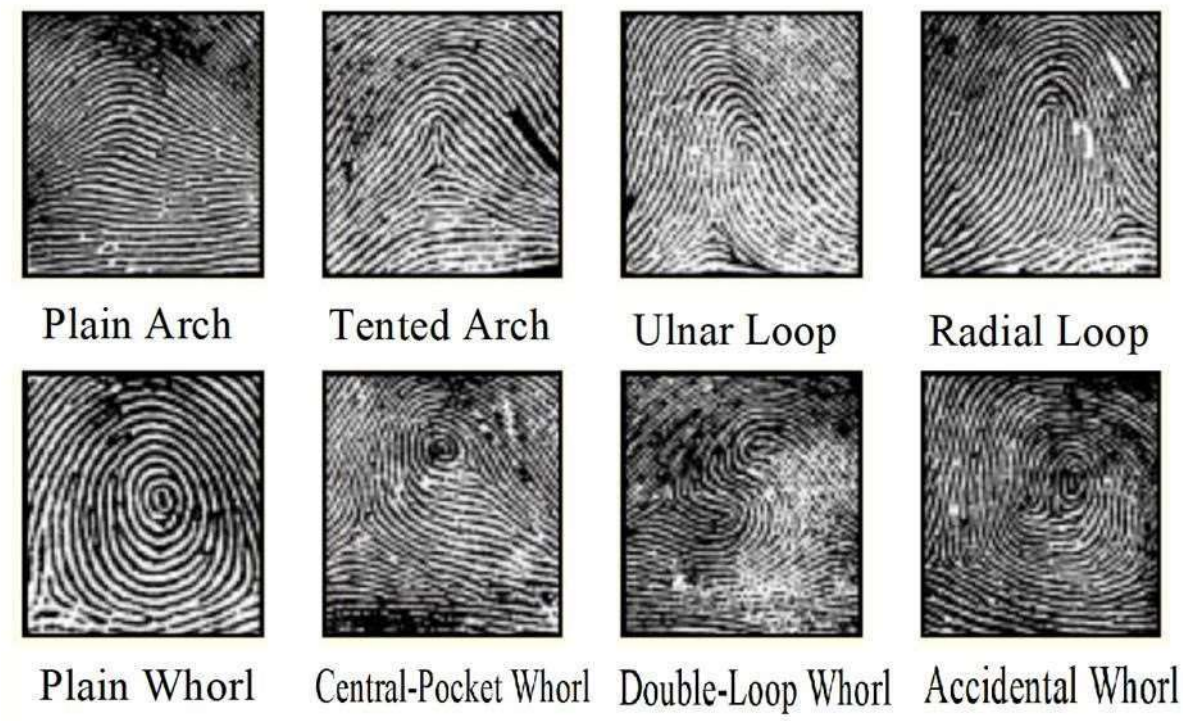

Figure 3-2 Henry's Classification of Fingerprint Patterns

For many years, research on fingerprint recognition has been done and as a result, many algorithms have been presented to make this system more flawless (Kai et al., 2012; Sebastian, 2014), most of suggestions shows that the finger should be held vertically which is more flawed for devices that are held in an alternate direction. These prints are stored as data in the form of pictures and can be used for authentication and/or identification. The majority of the research found in the literature on fingerprint recognition systems is based on the minutiae of fingerprints (Jain, Prabhakar, Hong, & Pankanti, 2001; Tan, & Bhanu, 2002, 2006) due to differentiation and unchanging for fingerprint, which makes each fingerprint unique. This differentiation is defined by minutiae, a collection of local and global characteristics such as ridge ends and ridge bifurcations and valleys and ridges (Kalogridis, Efthymiou, Denic, Lewis, Cepeda, 2010; Sruthy, 2013). (As shown in Figure 3-3).

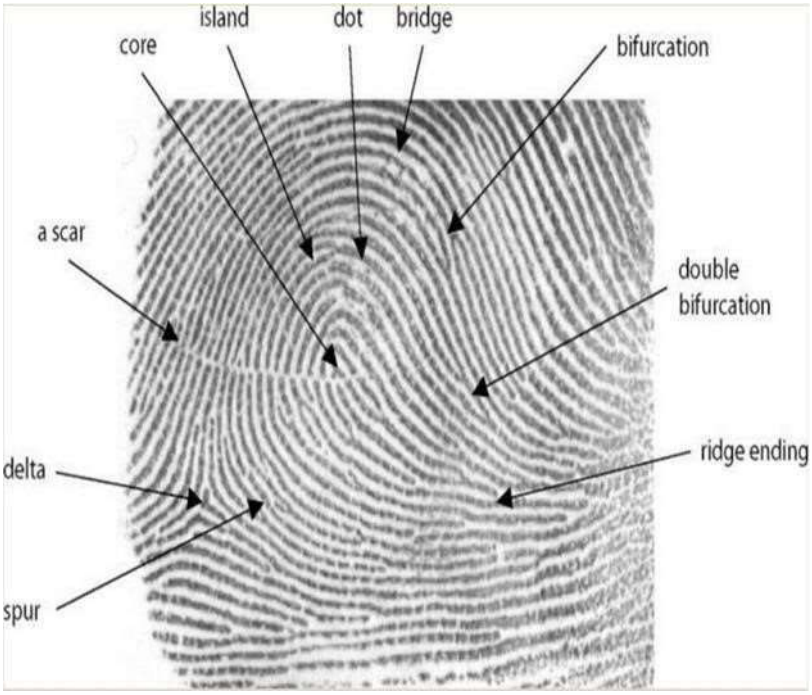

Figure 3-3 Fingerprint Minutiae (Sruthy, 2013; Kalogridis, Efthymiou, Denic, Lewis, Cepeda, 2010)

The problem I'll be dealing with in this chapter is properly characterised as follows.

3.3 Problem Definition

The increasing focus on fingerprint identification system research has improved the software and algorithms that underpin them. Many but not-enough algorithmic efforts have been made at accurate fingerprint pattern processing and interpretation. The disadvantage of this method was that it was costly, took up a lot of space, and was not suitable for large data. Also, the correctness of the alignment within record time was tough due to the complications of matching strings throughout the grid and modifying edit distances. As a result, this algorithm's accuracy and speed become mitigating variables. This leaves room for improvement.

Several commercial and proprietary systems exist in similar areas; however, the difficulty with the systems on the market is the sensors utilised to capture the fingerprint picture. Most biometric systems, in particular, make the unreasonable assumption that a fingerprint image is analyzed using the same sensor, limiting their capacity to match or evaluate biometric data from multiple sensors (Nguyen, Wang, & Li, 2013) (As shown in Figure 3-4).

Figure 3-4 Two fingerprint images of the same finger with different skin conditions acquired with a AFI scanners.

As seen in Figure 3-4, different scanners have been used for the input of a same fingerprint, which shows a lot of distortion, twisted ridge structures and interrupted minutiae points. To explain this further, contrasting warping modes will be obtained if the fingerprint is captured with different contact centres, and the fingerprint will be distorted due to the nonorthogonal pressure applied on the sensor (Nguyen, Wang, & Li, 2013).

Furthermore, some of the difficulties experienced in researching this extensively explored topic include, but are not restricted to, applicability in comparing biological systems.

The process for an Automated Fingerprint Identification System (AFIS) greatly depends on the accuracy of the minutiae extraction (Sebastian, 2013). The majority of AFIS measure minutiae data (such as bifurcation position and ridge endings) to identify and authenticate via sets of coordinates. However, not many studies have been focused on alternate methods available for minutiae extraction. In this chapter, a different method for fingerprint recognition has been proposed keeping in mind that minutiae extraction will be obtained as circular strings, appropriate for estimated circular string matching. Also, the suggested method can identify the exact point and rotation of the input fingerprint regardless of its input on the scanner. Our objective is to improve the minutiae extraction speed as far as possible and enhance the precision of minutiae extraction more efficiently. The suggested implementation code will have the ability to be coded directly in the mobile application.

3.3.1 Preliminaries

In Chapter 2, I went through Circular String pattern matching in great depth. I'm simply going through the fundamentals in terms of fingerprinting, where I have to utilize binary alphabet rather than genomic alphabet since the supplied circular strings are binary alphabet 0 and 1. A typical linear string with the left-most and right-most symbols looped around and fastened together may be seen as a circular string of length m. The same circular thread might be perceived as m separate linear strings, all of which are deemed comparable under this theory. I define x = x [0, 1,..., m1] as any circular string of length m given a string x of length m. In order to match the retrieved circular strings with the string renderings of the fingerprints recorded in the database independent of the various rotations that are present, we know that xi=x [i+1, I m, 0, I 1] is the ith rotation of the x pattern. xi=x I I + 1,.., m, 0, I 1] is the ith rotation of the x pattern. As a result, the approach's complexity is O(n); the Estimated Circular String-Matching technique (Bohli, Sorge, & Ugus, 2010) is used to determine the fingerprint's orientation. In my method, I use the binary alphabet, where each letter of the alphabet is represented by a numeric value of = 0 or 1. To represent the numeric value of the letter x, I use num(x), x. So, if the edge of a circle crosses with a ridge, I have num(0) = 0, and when it intersects with a furrow, I get num(1) = 1. The fingerprint picture fi is translated into s concentric circles, which may be converted into s circular binary strings.

If x = x[0, 1, .., m−1] is any circular string of length m.

X^i = x[i, i + 1, .., m, 0, 1, ..i − 1] is the ith rotation of x.

3.4 Contribution

The Needleman-Wunsch Landau, and Vishkin algorithms (Landau & Vishkin, 1998) are essential algorithms in this chapter because they solve string matching more precisely in record time while using less space at a low cost. The strategy employed suffix trees to cover more ground in less time. It accounts for the possibility of mistakes or mismatches caused by extraneous characters in both texts and patterns, as well as insertions and deletions (k differences).

The algorithms study the text via a 'window,' connected to the left end of the text and the window. The pattern is then compared against the text inside the window for a match, yielding a positive or negative result. An effort is a term for this procedure. The following text step is to test for a match or a mismatch once again. The sliding window system (Christian & Lecroq, 2004) is a method that repeats this operation till the full text is read.

The ability to recognise the precise position and rotation of the fingerprint will be supplied independently of the finger's position on the scanner, thanks to the suggested method in this chapter. [As shown in Figure 3-5 for an example.] Furthermore, by concentrating on increasing extraction speed, our goal is to increase efficiency to complete the whole process in less time.

When the fingerprint image is stored in various positions using circular string-matching algorithms to complete the matching procedure, it can deal with topographical error and thus allow the minutiae identification procedure to be completed in linear time according to the collective length of each searched string, this is accomplished through an initial orientation identification stage.

Fast Minutiae Extraction will generate circular strings for circular matching operations at this stage. (Section 3.4.1) goes into great detail about this, accompanied by an explanation of our method.

This was the goal of our method, which employed acdm/asmf (Advanced System Management Function/Advanced Collaborative Decision Making Function) to examine if the relative rotation of the circular string of the input matches the stored fingerprint image's circular string rotation. In addition, the Needleman-Wunsch approximation procedure assessed the alignment's correctness. Extracting minute details from an image and forming a circular string is a key factor in speeding up implementation since it is used repeatedly throughout the whole process.

The goal of the approach we provide in this chapter is to improve the accuracy of fingerprint identification by extracting more minutiae. The following two sections explain the suggested algorithm:

- Fast Minutiae Extraction and
- Our algorithm outlines

3.4.1 Fast Minutiae Extraction

The reason a lot of focus is given to minutiae extractions is that the efficiency of the extraction has an imperative effect on the whole time required for fingerprint recognition to be done effectively. A new extraction process known as Fast Minutiae Extraction is proposed in this chapter compared the results to Novel Minutiae Extraction-2 (Ajala, Iliopoulos, & Khan, 2016) that solve the same problem, which turns out to be much faster in all database size cases in my algorithm (Alshammary, Iliopoulos, & Khan, 2020).

Using our approach, database formation, a string can be formed by capturing a set of scan circles, accurate location and input fingerprint rotation. These data are saved as images in the fingerprint database, which can be matched afterwards. The major challenge is the position of the finger on the scanner. In the absence of correct orientation, it is impossible to compare with the image of a fingerprint in the database; the result will either be imprecise or nonviable. Hence, the purpose of data formation is to extract the circular strings to employ them to identify the fingerprint with the relative orientation. The precise location and orientation of the input fingerprint are achieved by extracting the circular strings, and the obtained data will be the fingerprint location in case there is a match. The rotation is used to adjust the input of the fingerprint images as per the matched image in the database. (As shown in Figure 3-5).

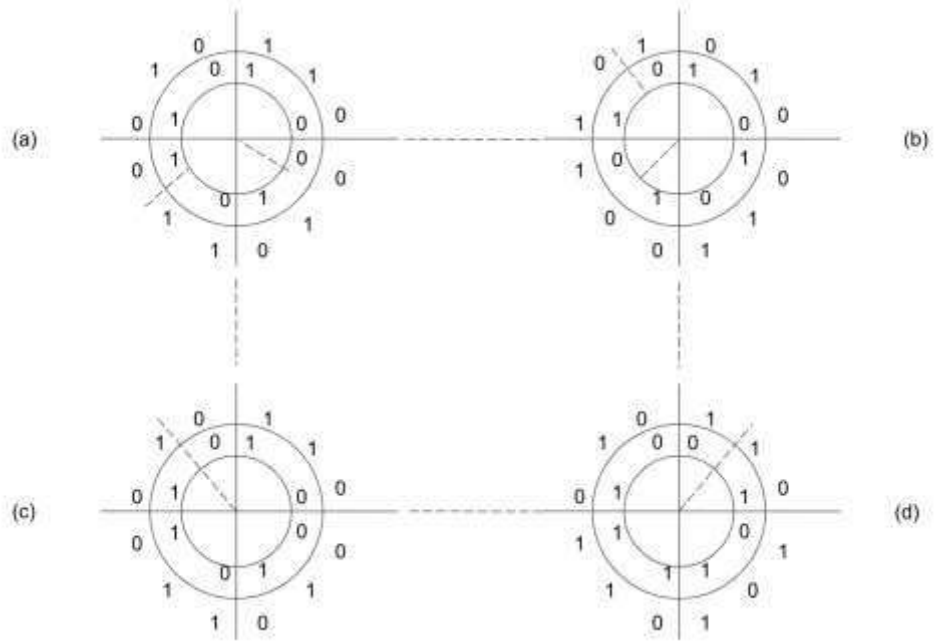

Figure 3-5 Extracting concentric circles and the corresponding binary strings of four arbitrary images

For the sake of clarity, just two circles are displayed here. A revolution of the outer circle in figure IV-C matches a revolution of the inner circle in figure IV-D, and both circles have the same degree of rotation in figure IV-C. (as shown by the dotted lines). A fingerprint's circular strings must be rotated at the same angle to recognise a picture as a fingerprint rotation. We extract the most extended circular strings from the input and the stored fingerprint. We use Advanced System Management Function/Advanced Collaborative Decision Making Function (asmf/acdmf) and Needleman-approximation Wunsch's approach to see whether any rotation of the circular string from the input fits the one from the stored fingerprint (or vice versa). The chances of the circular strings precisely matching are reduced because the input picture is rotated and distorted. Consequently, we may expect a more reliable outcome if we rely on approximation matching.

However, circular string is extracted individually by the extraction algorithm, this is achieved by obtaining the minutiae data by inputting a series of scan circles of a fingerprint. These data are then converted into strings consisting of 0s and 1s. We acquire the picture for each circular minutiae feature by inserting a 0 at the border of a circle that coincides with a ridge and a 1 at the edge of a circle that connects with a furrow [As shown in Figure 3-6].

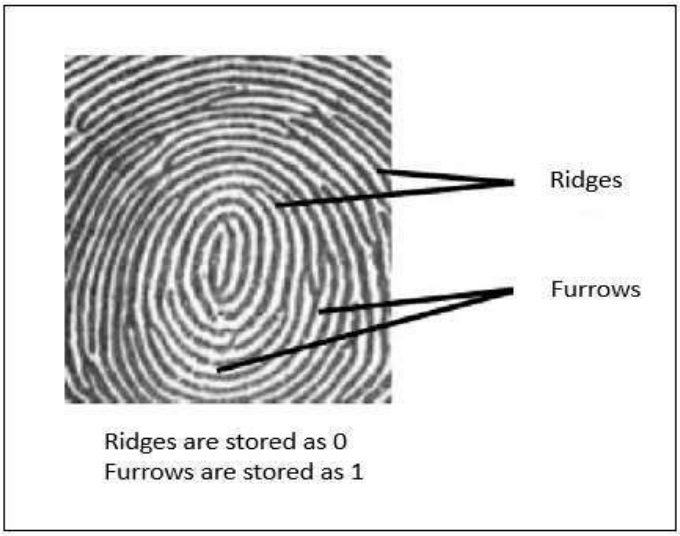

Figure 3-6 Ridges and Furrows on a fingerprint image

That means that the algorithm takes place S times, where S is the number of inner circles for every set (each focal point). An input parameter is a maximum radius employed to calculate the number of inner circles present in each set. Using the minimal radius and assuring that the distance across each circle at a comparable focus point is set [As shown in Figure 3-5], the number of inner circles may be computed. The result is a CsR list of radius pairs and circular strings (CStr, Radius>). To the length of the CStr, the sorted items are placed in decreasing order.

Cir is the circular string of the ith picture with a radius of r, and each spin in this iteration of the circular string is kept in the database. Allowing DbRir to be a series of strings from Cir; for example, if Cir = '0111,' DRir would have the values '0111,'1110,'1101,'1011'. In contrast, if DbRir is the collection of rotated strings from Cir; Cir ='0111', DRir will have the values '0111', '1110', '1101', and '1011'.

Moreover, The Approximate Circular String-Matching Algorithms (Bohli, Sorge, & Ugus, 2010) were used to search every time a rotation of the pattern takes place that has a length m in a text of length n. so as to identity the string representations of the fingerprints with the extracted circular strings saved in the database in linear time, increasing the efficiency and accuracy of the recognition method. The distortion of the input image was also greatly reduced because, rather than mirroring the minutiae features, our method focuses on approximation of the features rather than focusing on the static features of the minutiae that limits the print to vertical, we proposed focusing on circular strings, in which case the speed of the complete recognition feature is largely affected by extraction speed to some extent. In order to make this process more efficient and increase the extraction speed, our objective is to enhance the extraction procedure.

In Section 3.4.2, The following is the outline of Fast Minutae Extraction:

3.4.2 Outline of the Algorithm

ALGORITHM Fast_Minutae_Extraction

```
1: ALGORITHM Fast_Minutae_Extraction(char [][] img, int r, int cx, int cy)
2: INPUT: "img", a 2d char array representing a fingerprint
3: INPUT: "r", the radius of the circular string to be extracted
4: INPUT: (cx, cy), the centre of the circular string to be extracted
5: OUTPUT: "pattern", a circular binary string
6:      {
7:         string topLeft;
8:         string topRight;
9:         string bottomRight;
10:        string bottomLeft;
11:    for(int i = (cx -r); i < cx; i++)
12:     {
13:        double dJ = Math.Sqrt(Math.Pow(r, 2) – Math.Pow((i-cx),2));
14:        if(dJ == (int)dJ)
15:        {
16:              int xOffset = cx -i;
17:               int yOffset = (int)dJ;
18:               if(pixel at img [cx - xOffset] [cy + yOffset] < 125) {
19:                     Append "0" to the end of topLeft;
20:               } else {
21:                     Append "1" to the end of topLeft;
22:               }
23:                if(pixel at img [cx + yOffset] [cy + xOffset] < 125) {
24:                     Append "0" to the end of topRight;
25:               } else {
26:                     Append "1" to the end of topRight;
27:               }
28:                if(pixel at img [cx + xOffset] [cy – yOffset] < 125) {
29:                      Append "0" to the end of bottomRight;
30:               } else {
```

```
ALGORITHM Fast_Minutae_Extraction
31:                     Append "1" to the end of bottomRight;
32:               }
33:               if(pixel at img [cx - yOffset] [cy – xOffset] < 125) {
34:                     Append "0" to the end of bottomLeft;
35:               } else {
36:                     Append "1" to the end of bottomLeft;
37:               }
38:         }
39:   }
40:    return topLeft + topRight + bottomRight + bottomLeft;
41:   }
```

It executes Operation Fast Minutiae Extraction using (char [][] img, int r, int cx, int cy) as inputs parameter for each image of the supplied fingerprint (fi). Where image is a 2d char array to seek for the converging of circle C with radius r, with the centre at (cx, cy), which is sufficient to look for all (x, y) coordinates in compliance with the circumference of a circle equation: r = (cxr) to (cx). The method takes O(m) + O(n-m) = O time to execute (n).

This approach produces a single circular binary string retrieved from a circular template with a radius of r and a center point (cx, cy). It's worth noting that the generated string has alphabet = 0,1 in this example. x′ is a function of each rotation of x. We obtain a 0 at each edge of a circle that meets with a ridge and a 1 at every edge of a circle that connects with a furrow for every circular microscopic element of the picture.

To explain the algorithm in detail, lines 7-10 defined four strings as:

- topLeft
- topRight
- bottomLeft
- bottomRight

For each fingerprint image provided as an input will divide into 4 equal halves in 4 areas [As shown in Figure 3-8], the scan is then performed on each region. To explain it more, for example, the fingerprint image is 512px wide and 480px high [as shown in Figure 3-7]. The image will be split in 4 equal halves [as shown in Figure 3-8].

Figure 3-7 Fingerprint image from the database

Each of these halves will be 256px wide and 240px high. This process is done to search specific regions of the fingerprint.

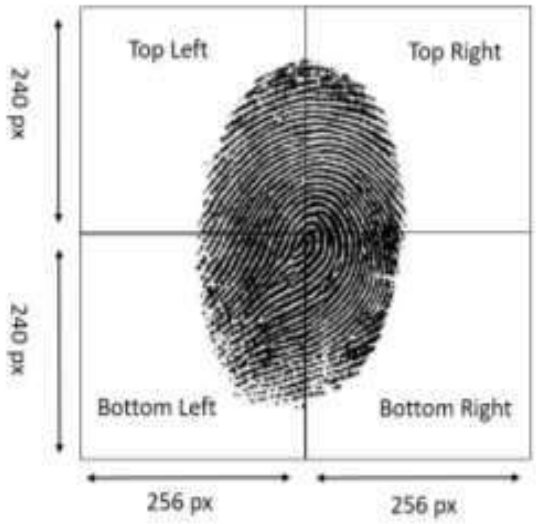

Figure 3-8 Fingerprint Divided into 4 blocks

Line 12 starts a loop that runs for each fingerprint picture entered. a simple loop in Fast Minutiae Extraction that runs r times (from (cx-r) to cx) instead of 2r (from (cx-r) to (cx+r)) in Novel Minutiae Extraction-2 (Ajala, Iliopoulos, & Khan, 2016), which means the total number of sqrt (r^2 - (i-cx) ^ 2) evaluations will be halved. The method takes O(m) + O(n-m) = O time to execute (n). Instead of scanning the whole fingerprint scan, this will allow you to search simply the area where the scan circles are extracted. Consequently, the algorithm's efficiency has grown by orders of magnitude. [As shown in Figure 3-11]

In lines 13 to 26, the algorithm returns a list of fingerprints from the database (DB). Each with a rank. The one that matches most with the input has the highest rank [As shown in Figure 3-11]. Line 14 defines a Double data-type variable, dJ. By making use of a mathematical formula: sqrt (r^2 - (i-cx) ^ 2), the total number of evaluations are obtained. As a result, the system may search just a specified location rather than the whole fingerprint scan. Thereby increasing the efficiency. Moreover, the Double data type used in the variable “dJ” can hold a precise value.

In the following line, i.e., Line number 15, the comparison is made between the string obtained and the strings of fingerprint images already existing in the database. This line at the bottom gives a rank-ordered list of fingerprints from the database. The one with the greatest rank [As shown in Figure 3-11] is the one that most closely matches the input.

This step might be repeated; the only issue now is determining the best match among the circles of the same radius. To do so, we utilise the Estimated Circular String-Matching with the Needleman-Wunsch algorithm (Landau & Vishkin, 1998) to check the precision of the alignment and the Advanced System Management Function/Advanced Collaborative Decision Making Function (asmf/acdmf) technique to ensure that the spin of the circle string of the input matched that of the stored fingerprint picture, which takes time O. (n). Landau and Vishkin (1998) contains comprehensive information on how the Needleman-Wunsch algorithms function.

In the line numbers 20, 26, 32, and 38, the previous equation is applied, and offsets of the fingerprint are calculated. This process is done to identify the boundary of the input image, even if it has been rotated [As shown in Figure 3-9]. For example, if we rotate the saved image by 90° degrees or by 180° degrees, we roughly 1500 points on the input image who could be a potential centre of the fingerprint. As part of my test I have rotated the image by various degrees and counted the number of potential centres for each case that shows how the number of prospective centres changes with the rotation of the input image. The suitability of allowing errors in locating occurrences (i.e., approximate version) rather than considering exact occurrences (i.e., exact version) remains true in the context of circular pattern matching as well. Therefore, the biologists are more interested in locating the approximate occurrences of one of the rotations of P in T, i.e., they like to allow errors while locating the occurrences. This motivates one to study CPM.

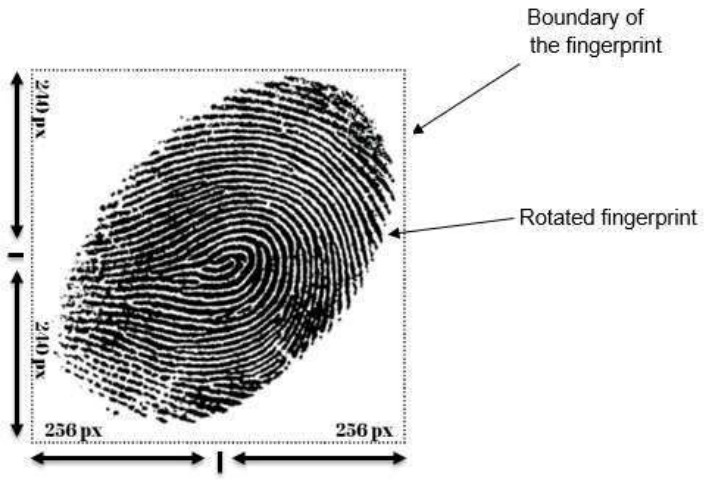

Figure 3-9 Fingerprint boundary of a rotated input image

The algorithm at lines 20, 26, 32, and 38 performs a check on each of the 4 split regions discussed previously. Then, in each of these "If-else" statements, the algorithm identifies the pixels that intersect the ridges and furrows of the fingerprint. If the pixel lies on the ridge, it is stored as a 0 in the binary string, and if the pixel hits the furrow, 1 is stored. Because pixels are recorded as integers with values ranging from 0 to 255, it is necessary to verify whether the value is less than or larger than 125 to determine whether the pixel connects with a ridge or a furrow.

It is now able to validate the input fingerprint using typical string-matching algorithms for comparing the input fingerprint to the fingerprint picture previously preserved in the database; To identify an array of candidate matches on the scanner surface, we used these aids to point to the 'candidate image' and then the (i+1) th image until all the images in the database were used, and then gives the minutiae (in the form of circular strings), such as their location on the scanner surface and the best alignment, i.e. what kind of rotation to use for re-alignment of the fingerprint. When all the strings are linked together, the match pictures are returned at line 45.

The primary objective is to reduce the number of potential candidates to a manageable amount. Upon completion of the procedure, it is presumed that there will be two pictures of replicated size and/or position that must be matched and checked, from which some pairing has already been obtained. The following steps will be taken to provide clarity as we go through the present stage. You can easily convert either picture to a one-dimensional binary string by changing the two-dimensional binary nexus between the images to a zero/one value nexus. Pattern matching between strands of the same length is the sole remaining duty. However, it must be kept in mind that there is a chance for mistakes to occur. The fingerprint may be identified, we compute and modify the distance between the two binary strands to be within the permissible tolerance threshold.

3.5 Experimental Results

The experiments of Fast Minutiae Extraction (Alshammary, Iliopoulos, & Khan, 2020) were conducted in C# programming language on a Desktop PC using an Intel Pentium processor (CPU 4415U) with 2.30 GHz, 4.00 GB of RAM, 64-bit Windows 10, and the MS.Net Framework 4.5 platform was utilised for the proposed solution.

The experiment has been tested with 8 bit Gray-level BMP files and image resolution 328*356, where are captured from the CASIA fingerprint biometric database version 5.0 (or CASIAFingerprintV5) contains 20,000 fingerprint images of 500 subjects. The fingerprint images of CASIA-FingerprintV5 were captured using URU4000 fingerprint sensor in one session (CASIA Fingerprint databases, 2020).

According to the experiments results, Minutiae is separated into circular strings of data rather than employing the ridge endpoints or bifurcations as fingerprint identifying elements, as is the case in the proposed approach. Minutiae separation results in a reduction in overall execution time due to increased performance. To ensure that each of the occurrences of the pattern of length m within a text of length n, matches the stored fingerprint image, we use the Aproximate Circular Pattern Matching, Needleman-Wunsch algorithm (Landau & Vishkin, 1998) to calculate the alignment's accuracy and precision, and Advanced System Management Function/Advanced Collaborative Decision Making Function (asmf/acdmf) technique ensures that each circular string's rotation is inputted data that matches the stored fingerprint image.

A comparison will be made between the fingerprint submitted and another fingerprint stored in the database, and the results will be as follows: When the device boots up, two fingerprint pictures are loaded twice, in a 2dimensional arrangement of pixels. The number one picture is the scanner's entered fingerprint, and the number two image is the database's stored fingerprint, assuming both images are using the same scanner standards (As shown in Figure 3-10).

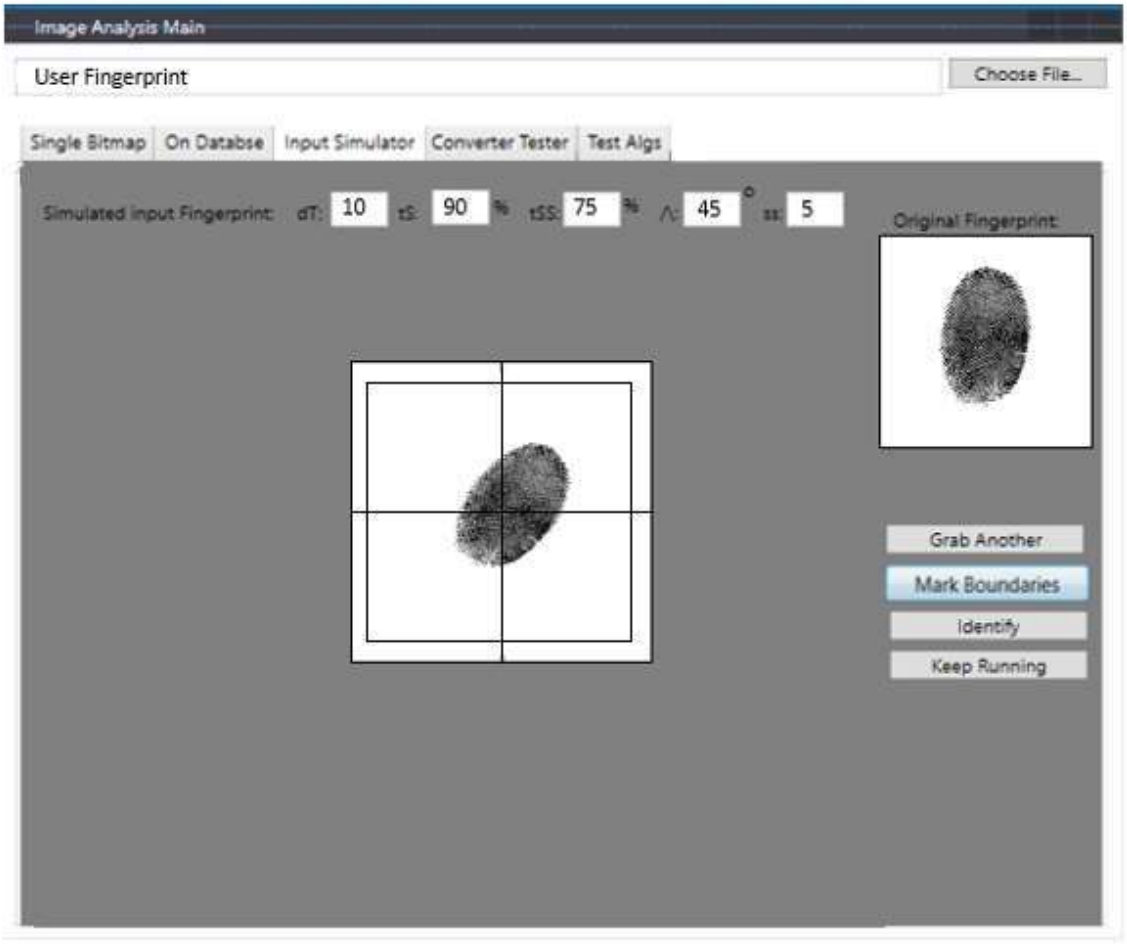

Figure 3-10 Input panel with a simulated input and its identified image-boundary

Figure 3-10 shows the input panel with fingerprint image 512px wide and 480px high, that have been rotated by rotation degree 45 inside the input panel with a high specific threshold (tS), tS=90.

The proposed algorithm separate minutiae details from the input fingerprint image and forms the circular string, the percentage of matched strings are compared with threshold (tS), if it's higher then will mark the ith image as a candidate and select the next (i + 1)th image. This process is repeated until all entered fingerprint images are matched.

The performance and accuracy of the proposed algorithm depends on the size of the database as well as on the potential centre of the fingerprint on the input image. This number varies depending on the rotation of the input image. As part of my test, I have rotated the image by various degrees and counted the number of potential centres for each case that shows how the number of prospective centres changes with the rotation of the input image. For example, if we rotate the saved image by 60° degrees or by 240° degrees, we roughly 30,000 points on the input image who could be a potential centre of the fingerprint. Also, if we rotate the saved image by 45° degrees or by 225° degrees, we roughly 40,000 points and so on. This, in turn, greatly enhances execution performance.

The extracting time and matching time were not affected with the rotation degree nor the accuracy of the results when Scanning and Matching same Fingerprint with different rotation degree.

I have implemented Fast Minutiae Extraction and conducted extensive experiments to analyse its performance. I show the comparison based on the experimental result between Fast Minutiae Extraction algorithm (Alshammary, Iliopoulos, & Khan, 2020) and the algorithm of Novel Minutiae Extraction-2 (Ajala, Iliopoulos, & Khan, 2016) that solve the same problem, which turns out to be much faster in all database size cases in Fast Minutiae Extraction algorithm. Figure 3-11 reports the elapsed time and speed-up comparisons for various database sizes. As can be seen from Figure 3-11, the Fast Minutiae Extraction algorithm runs faster than Novel Minutiae Extraction2 in all cases.

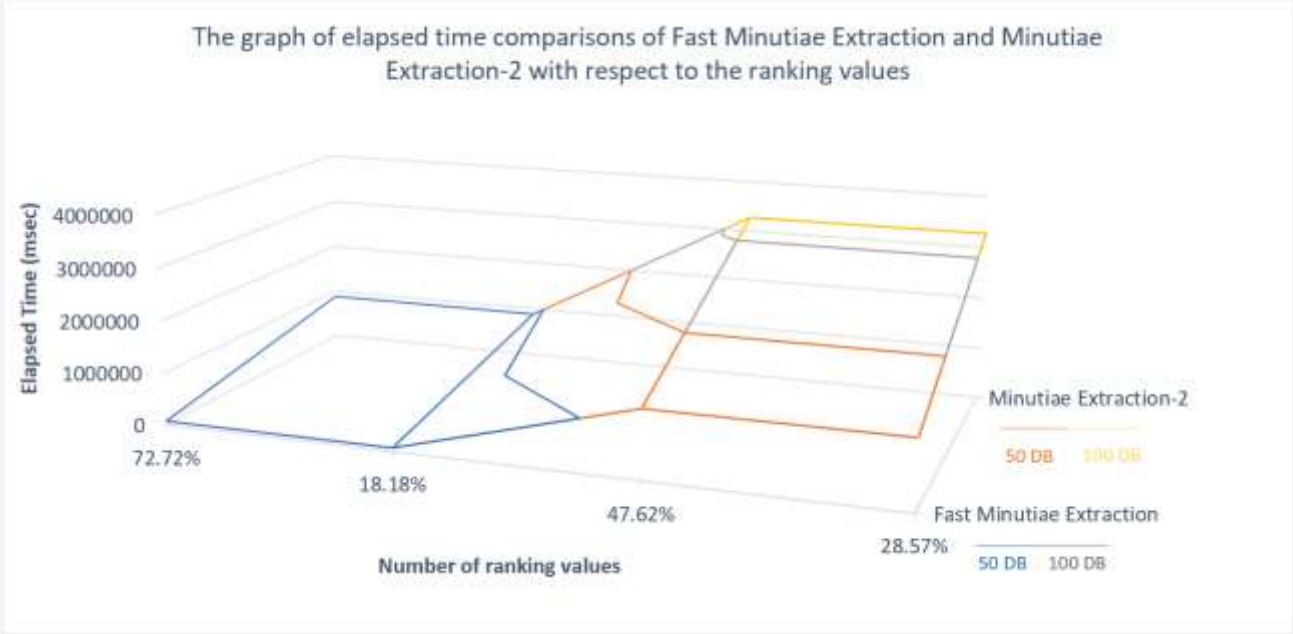

Figure 3-11: Elapsed-time comparisons of Fast Minutiae Extraction and Minutiae Extraction-2.

The figure 3-11 above shows the elapsed-time comparisons between Fast Minutiae Extraction algorithm and Minutiae Extraction-2 algorithm with respect to the ranking values in different database cases, where is the size of database is the number of stored images and the input fingerprint is matched against them. The algorithm returns a list of fingerprints from the database each with a ranking value, the one matches most with the input has the highest rank in each database case. The highest percentage of each database cases is the correct fingerprint identified as the top ranked match, that means the algorithm identified the correct fingerprint. The lowest percentage of each database cases is where the algorithm fails to rank the correct fingerprint in the list of top 5. As can be seen from Figure 3-11, the Fast Minutiae Extraction algorithm runs faster than Novel Minutiae Extraction-2 in all cases.

The proposed solution's flow chart is shown in Figure 3-11.

3.5.1 Time Complexity:

As the BitMap is only looped through once, the time complexity is linear with O(n), and a character array is built by combining 8 bits and converting them to ASCII, which occurs only once in 1/64 of a byte (1 byte is equivalent to 64 bits).

The linear temporal complexity is O (n+n/64), O (n^2) is the space complexity in our algorithm.

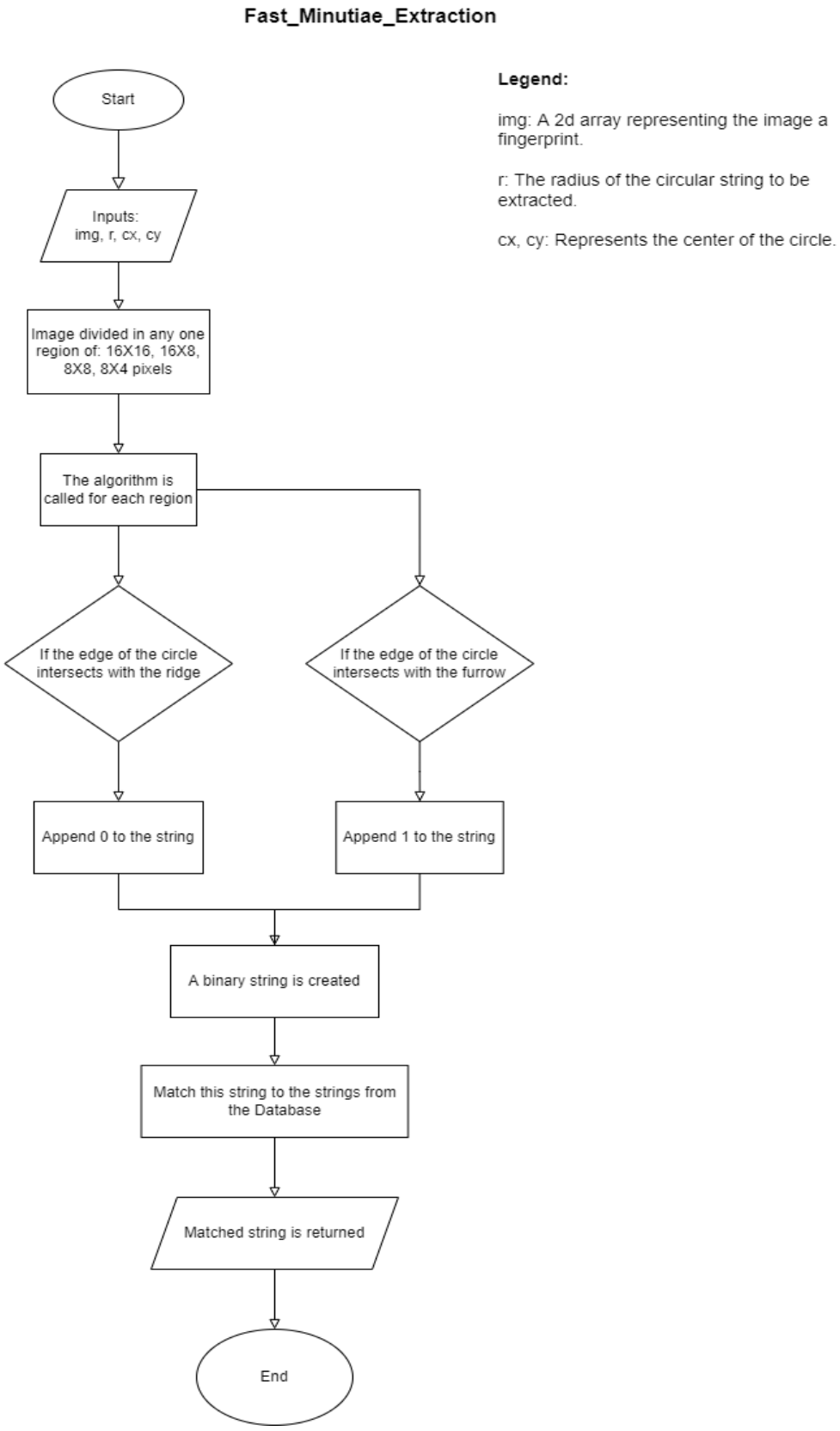

Figure 3-12 Flow chart of the proposed solution, the fingerprint identification system

3.6 Discussion and Future Work

As fingerprint recognition systems have become increasingly adopted within a range of technology applications over the last decade, so too has their attention within emerging research. However, although this increased attention has led to an enhancement of the software and algorithms behind this recognition process, but there is still room for improvement.

A new extraction process known as Fast Minutiae Extraction algorithm is proposed in this chapter, for the approximate circular string pattern matching problem of fingerprints recognition system- based on mobile device identification. The experimental results are compared with previous studies which solve the same problem (finger orientations), the results showing that the proposed algorithm is much faster in all cases of different database sizes, due to the massive reduction in search space, which running in linear time, O(n).

In future work, the algorithm will be expanded to include several different types of databases in recognition system besides fingerprint such as face print ...etc.

A face print or face recognition is very similar to fingerprint recognition using a circular string pattern matching algorithm, and the same mechanism works from detecting rotation to entering an image and correcting it using a set of operations implemented - to extraction points.

For example, in face recognition system, nodal points are end points used to measure variables of a person's face, such as the length or width of the nose, the depth of the eye sockets and the shape of the cheekbones. The system works by capturing data for nodal points on a digital image of an individual's face and storing the resulting data as a faceprint. The faceprint is then used as a basis for comparison with the data captured from faces in an image or video. It is quite similar to how minutiae extraction works in a fingerprint recognition system where the minutiae points are detected by locating the end points and bifurcation points on the thinned ridge skeleton based on the number of neighboring pixels. The transitions from 0 to 1 or vice versa when the binary string is considered circular.

4 Application and Algorithm: Maximal Motif Discovery for Biological Data in a Sliding Window

This research study was published and presented at the 16th International Conference on Artificial Intelligence Applications and Innovations (Alshammary, Iliopoulos, Mohamed, & Vayani, 2020). Implementation and experimentation were the author's contributions to this work.

4.1 Introduction

With the advancement of next-generation sequencing technology, the amount of genomic data produced has increased, necessitating de novo assembly and analysis motif identification (Carvalho, Freitas, Oliveira, & Sagot, 2006; Sinha & Tompa, 2003; Pavesi, Mercghetti, Mauri, & Pesole, 2004; Pisanti, Carvalho, Marsan, & Sagot, 2006; Pissis, 2014; Pissis, Stamatakis, & Pavlidis, 2013). Motifs are short sequence patterns that often appear sufficiently in an alignment of identical or similar sequences to be deemed conserved, notwithstanding their biological significance. Motif discovery problem is a computationally challenging task (Pevzner & Sze, 2000), where the challenge is to identify the motifs, i.e., the sequence of letters that make up a given motif. However, because the implementation inputs of experiment for maximum motif identification in biological data in this chapter are circular strings, it was required to grasp and study the cyclic factors for strings in Chapter 2 because viruses are considered to be the inspiration for cyclic factors. The DNA sequence of many viruses has circular structures. To find occurrences of a particular virus in a carrier's (linear) DNA sequence, must locate all interesting circular patterns P that occurs in text or database T, and this defines a circular pattern discovery (CPD) problem. For basic examples of Escherichia coli (E. coli) and the splitting of the viruses in the circle, see Figures (2-3) and (2-4) in Chapter 2. See Figures (4-3) and (4-4) below for more complicated instances. The suffix tree and suffix link were presented in Chapter 2 that utilized as computing tools in this chapter were thoroughly specified.

Since binding sites have a biological purpose and are thus under selection pressure, they are better retained in DNA. We can use motif discovery techniques to assist us in finding them. This chapter seeks to locate all ℓ length motifs from a specified collection of sequences occurring in at least *k* sequences. Every motif appearance may have up to *d* mismatches (Grossi, Mermcorri, Pisanti, Trani & Vind 2018) using the procedures in (Iliopoulos, Mohamed, Pissis, & Vayani, 2018). One of the drawbacks of (ℓ,d)-motif search methods is the motif's limited length. No specific input pattern is provided, a pattern that is shorter or longer may be more meaningful. As a result, the emphasis of this chapter is on the broader topic of maximum motif finding and what is interesting is typically defined in terms of constraints in the search, where *k* is the minimum number of occurrences of maximal motifs and *d* is the maximum allowed error(s). In a practical context, such errors may arise due to various reasons ranging from natural mutations in the DNA of the virus to the practical limitations of the lab equipment's that may introduce errors while sequencing.

4.1.1 Chapter Summary

This chapter describes how the maximum motif finding method is used in bioinformatics to solve real-world genomic data issues. I present a Maximal Motif Discovery in the Sliding Window (MMDSW) algorithm, for the circular pattern discovery (CPD) problem that uses suffix trees as computing tools, where ℓ is the maximum length of the circular patterns and x is the total length of the sequence database. I utilised actual genomic data as input sequences, this information was gathered from the European Nucleotide Archive (ENA), which will divide into substrings, ℓ -length of window size, to identify a set $\widetilde{M}_{i,d,k}$, maximum motifs in an input string that occur at least *k* times in the window and include at most *d* 'don't care symbols.'

The rest of the chapter is organised as follows: Section 4.2 providing the background and reviewing related literature. Section 4.3 gives a problem definition and preliminary description along with the terminologies and concepts related to sliding window that will be used in this chapter. Section 4.4 presents the contribution. Section 4.5 presents the experimental results. A brief conclusion in Section 4.6.

4.2 Background and Related Work

The authors in (Iliopoulos, Mohamed, Solon, Vayani, 2018) proposed the first online algorithm for finding every occurrence of all maximal motifs in a sliding window, $\mathcal{O}(nd\ell + d\lceil\frac{\ell}{w}\rceil \cdot \sum_{i=\ell}^{n-1} |\Delta_{i-1}^{i}|)$ where w is the machine word size and $DIFF_{i-1}^{i}$ is the symmetric difference of the set of occurrences of maximal motifs at x [i−ℓ..i-1] and x[i−ℓ+1..i], The algorithm in this study has an $O(\ell^2)$ space complexity.

An overview of (ℓ,d)-motif search approaches been proposed (Eskin & Pevzner, 2002; Rigoutsos & Floratos, 2006a; Pisanti, Carvalho, Marsan, & Sagot, 2006b; Pevzner & Sze, 2000).

Using the suffix tree (Crochemore, Hancart, and Lecroq (2007), McCreight (1976), Ukkonen (1992), Weiner, (1973), and Gusfield (1997a) as computation tools to solve the maximal motif discovery problem.

Farach (1997) gave the first suffix tree construction algorithm that is optimal for all alphabets, Farach's algorithm has become the basis for new algorithms for constructing both suffix trees and suffix arrays, for example, in external memory, compressed, succinct, etc.

The primary incentive for creating a dynamic structure to permit a sliding window on the real input sequence was the implementation of bioinformatics applications (Alshammary, Iliopoulos, Mohamed, & Vayani, 2020). This, in particular, enables the discovery of intriguing ℓ -length sequence areas, which is particularly valuable in various bioinformatics applications, such as predicting the origin of chromosomal replication (OriC) (Meijer, Beck, Hansen, Bergmans, Messer, Von Meyenburg, and Schaller, 1979) to solve problem of maximal motif discovery in sliding window using the motif trie approach (Grossi; Menconi; Pisanti; Trani; and Vind, 2014; 2017b).

The research problem I will be dealing with in this chapter is properly characterised as follows.

4.3 Problem Definition

The motif discovery problem consists of finding over-represented patterns in a collection of bio-sequences. Formally, a motif is a sequence of letters of the alphabet and don't care letters. A motif $\widetilde{M}_{i,d,k}$ that occurs at least *k* times in a sequence is maximal if it cannot be extended (to the left or right).

Finding patterns in DNA sequences is a demanding process in terms of the calculative aspects (Jones & Pevzner, 2004; Zare-Mirakabad, Davoodi, Ahrabian, Nowzari-Dalini, Sadeghi, & Goliaei, 2009). The protection for binding sites resides inside the DNA since they serve a purpose in the biological functions of the body, it is necessary for them to be well protected and regulated.

Some pattern finding algorithms aid in detecting binding sites and knowing such information is an important step in comprehending how gene expression is achieved (Modan K Das, Ho-Kwok Dai, 2007). Finding the binding sites needs a model to act as a base which is also called a pattern/motif. It can be done by utilizing a word-based model established upon alphabets used by IUPAC (IUPAC-IUB Commission on Biochemical Nomenclature, 1971) Table 4-1.

Table 4-1: IUPAC alphabet.

Code	Description
A	Adenine
C	Cytosine
G	Guanine
T	Thymine
R	Purine {A, G}
Y	Pyrimidine (C, T, or U}
M	{C, A}
D	{A, G, T}
H	{A, C, T}
V	{A, C, G}
B	{C, G, T}
X, N or	Σ

Patterns are called maximal, because it is not possible to add any data either to the left or the right unless there is a decrease in the number of original sequences. It is important for the patterns to have a substantial quantity of sequences available, therefore initiating with the *k* specification grants the minimum requirement for the times of sequences in a given maximal pattern/motif. For patterns such as $M_{i,d,k,}$ residing within the ℓ-length part of the window terminating at position i, for the strand x, all the pieces one by one which have to appear *k* times in the window and the maximum number of 'don't care symbol' have to be d. The starting specification *k* gives the minimum requirement for the number of times the sequences can appear in a maximal pattern, whereas the succeeding specification d is stricter since incongruity can occur to d specific parts in the pattern also called 'don't care letters' written or typed as ⋄.

Accordingly, the pattern can be maximal, because its 'don't care letters' are unspecialized unless there is a decrease in the number of sequences.

$M_{i,\,d,\,k}$ denoting as the combination of patterns existing on the ℓ-length segment of the window ending at the position i, for the strand X, all the pieces must occur *K* times in the window one by one, and the maximum number of 'don't care symbol' must be *d*. All pieces must occur *K* times in the window, with the maximum number of 'don't care symbols' being *d*. The resultant is a strand bundle in SX, where SX[i] =$|M_{i,d,k}|$and $\ell \leq i < n$.

4.3.1 Preliminaries

Chapter 2 went through circular pattern discovery (CPD) and the great depth of the suffix tree utilised as computing tools. I'm simply going over the fundamentals here: the proposed existing algorithm works by add a string of ℓ letters not in Σ to the start of x, whilst also appending a unique $ letter not in Σ to the end of x to ensure that the motif graph (augmented suffix tree Tx) of the final window is explicit.

The algorithms look at the text via a 'window,' connected to the left end of the text and the window. The pattern is then compared against the text inside the window for a match, yielding a positive or negative result. An attempt is a term for this procedure. The following text step is to check for a match or a mismatch. The sliding window system (Christian & Lecroq, 2004) is a method that repeats this operation until the full text is read.

This Sliding Window mechanism is depicted in the diagram below:

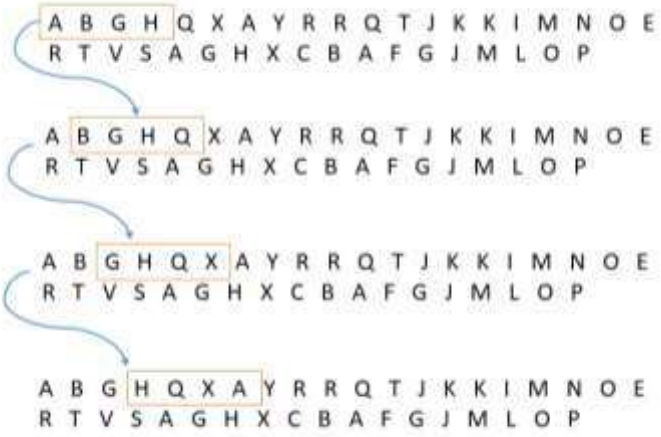

Figure 4-1 Sliding window mechanism

A string on the DNA sequence has been defined in the diagram above. The sliding window is assumed to consist of 4 strings. The window first selects the initial 4 values in a string, then moves one step ahead and selects the following 4 strings. The process continues until the entire string has been exhausted.

4.4 Contribution

The suffix tree is used in our proposed algorithm to discover these repeating regions in DNA sequences. This results in a thorough maximal motif discovery approach in a sliding window, where many motifs will be examined. The branch & bound technique is used to limit the candidate space. Especially when screening through vast amounts of biological sequences, Squandering combinations of letters like this, might be undesirable because mismatches in transcription factor binding sites can arise. This may be corrected by utilizing a suffix tree to find these repeating DNA sequence locations.

In this approach, the proposed solution sought to discover maximal motifs for all ℓ length motifs found in at least k sequences, with each motif occurrence containing up to d mismatches. A motif is a sequence of letters from the alphabet and letters that don't care. A maximum motif $\widetilde{M}_{d,\,k}$ appears at least k times in a sequence if it can't be expanded (to the left or right) or specialised (that is, its $d' \leq d$ don't care can't be substituted with letters from the alphabet) without diminishing its number of occurrences.

The sliding window method is applied in our proposed solution, reporting motifs was the biggest computational challenge from adding a letter to the right of the window, deleting a letter from the left of the window in the motif graph (As shown in Figure 4-2 and 4-3) and the set of motifs is updating in sliding window as follows:

1. Adding a Letter to the Right

Generally, adding a letter to the right in a trie is a basic approach and can quickly be done by inserting every key as an individual node in the trie. The children represent a cluster of points (or references) for the next level trie nodes. The important characters act as an index for the cluster children. Considering this, we use the input key, which might expand an existing key or a new key entirely. We would need to generate non-existing nodes of the key and record the end of the last word in the last node. Assuming that the input key is a prefix of the existing key in the trie, we would simply mark the last node of the key as the end of a word. Key span predicts trie depth.

2. Deleting a Letter from the Left

There is a simple rule to follow for deleting operations: to delete the key from the bottom in an upward direction utilizing the recursion technique. Subsequent outcomes can result when we delete a key from the trie.

- The key might not even exist previously in the trie, so there will be no modification to the sequence when deleting.
- The key might be unique (no part of a key contains another key (prefix), nor is the key itself a prefix of another key in a trie). The result will be the deletion of all the nodes.
- The key acts as a prefix key for another longer key in the trie. The outcome will be unmarking the leaf node.
- The key is present in a trie, having at least one other key as a prefix key. Delete nodes from the end of the key until the first leaf node of the longest prefix key.

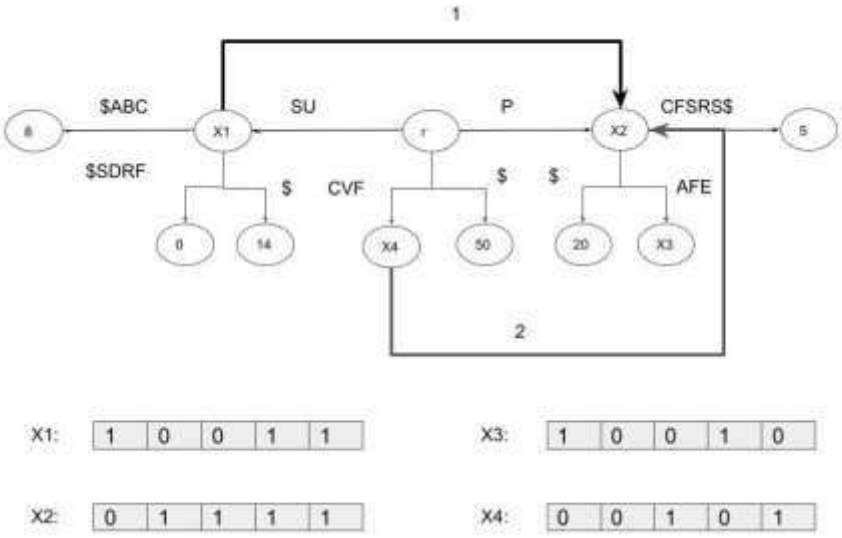

Figure 4-2: The motif graph for the suffix tree of the string X =SRFSRSUPACB$, given d = 1 and k <= 2. Each leaf node has been labelled with the index i of the suffix X [i... n−1] that it represents, where i ∈ (0; n). The set of all internal nodes, {X1 - X4}, represents the right maximal repeated factors of x. As |occ(x4) | = 4 = |occ(x1) |, x4 is not a seed; as |occ(x1) | = 4 = |occ(x4) | = 4, x1 is a seed, and so on. Motif edges and their labels are bold, all other explicit nodes are labelled X and suffix links are shown as regular lines.

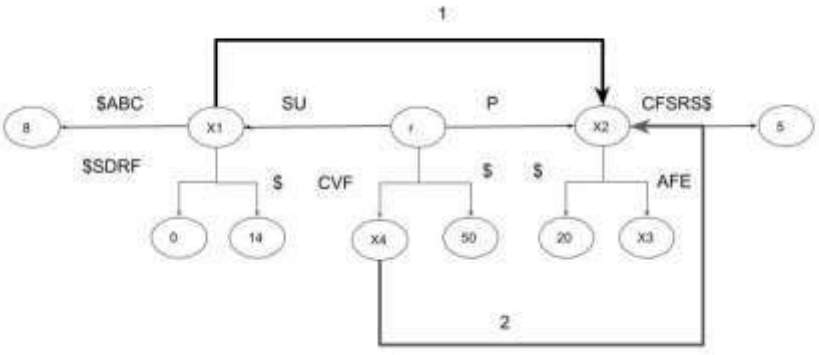

Figure 4-3: The implicit suffix tree of the string x = SRFSRSUPACB. Each leaf node has been labelled with the index of the suffix j that it represents for all suffixes j ∈ (0; 12). Observe that the most recent leaf to be added is suffix j−1 = 20. The active point is ‘r’ between x4 and node 50. The most extended repeated suffix x [12..19] = SRFSRSUPACB is the path from r, the root node to P, the edge corresponding to r. Motif edges and their labels are bold, all other explicit nodes are labelled X, and suffix links are shown as regular lines.

The benefit of employing an implicit suffix tree in our execution approach is that it takes up less space than the original suffix tree since it has fewer leaves. This is due to the excessive complexity of the naive suffix tree construction. In particular, points known as suffix links speed up tree traverse. Edge-label compression saves memory, and the leaf update rule is quicker when avoiding duplicate insertions in the tree, which speeds up other operations.

The suffix links are pointers connecting each internal node of an (implicit) suffix tree. They allow for a reduced temporal complexity O(n), linear time and space. This contributes to the needed the linear-time algorithm's increased speed. Suffix linkages also save time while traversing the tree from the root to the suffixes of S, allowing one of many seminal algorithms to create the suffix tree S(x) in O(n) time and space.

Additionally, we aim to utilize applications from bioinformatics to decrease the time of execution for solving such problems by increasing its efficiency. The coding I created is easily accessible since it can be integrated into a mobile application.

We introduce the special symbols used in our implementation in the Table 4-2.

Table 4-2 List of the Degenerate Symbols

Table of Symbols	
Number of sequences	X
Sequence length	n
Maximum pattern length	k
Pattern length |P|	|P|
Pattern	P
Motif alphabet	$\boldsymbol{\Sigma}$
Alphabet size	$\|\boldsymbol{\Sigma}\|$
Window length	ℓ
Internal Node	V
Motif	$\widetilde{M}$
New Motif	$\widetilde{M}'$
Nodes of the tree	M_i

Table 4-2 shows these special symbols, known as degenerate symbols that each represent a nonempty subset of the DNA alphabet, Σ = {A, C, G, T}

4.4.1 Outline of the Algorithm

The Algorithm in pseudocode:

ALGORITHM Maximal Motif Discovery for Biological Data in a Sliding Window

1: ALGORITHM MAXIMAL MOTIF DISCOVERY IN A SLIDING WINDOW (String X, int windowSize ℓ, int minThreshold, int mismatchOccur)

2: INPUT: "X", a String as the complete genome sequence.

3: INPUT: "ℓ" WindowSize, a substring extracted from the string X.

4: INPUT: "d" mismatchOccur as entered by the user - The maximum number d of allowed don't care symbols, Mismatch.

5: INPUT: "K" minThreshold as entered by the user, the minimum number k of occurrences of maximal motifs, Match.

6: OUTPUT: An array SX of motifs, where SX[i] = $|M_{i,d,k}|$ and $\ell \leq i < n$.

```
7: Suffix_tree{
8: Current node <- do current add function(new node function);
9: Constructor function Suffix_tree(argument x) {
10: for each i ε string length
11: root <- insertSuffix(argument x.substring(indexer), argument indexer);
12: boolean function check (argument one, argument two)
13:    {
14:       Variable n <-  listsize;
15:         // If the array is empty
16:            if n == 0
17:            then
18:            return false;
19:       for each i ε string length and i < n
20:       function Right_Handside(Argument x){
21:       if (function x->find(ℓ) >= k)
```

ALGORITHM Maximal Motif Discovery for Biological Data in a Sliding Window

22: then, check conditions for

23: SX[i] <- substring(indexer);

24: SX[i] <- M_i [indexer];

25: SX[i] <- $\widetilde{M}$ [indexer];

26: return $\widetilde{M}'$;

27: }

28: function Left_Handside(Argument x){

29: if !diff, check conditions for

30: then

31: diff <- true;

32: M_i [indexer] <- null;

33: SX[indexer] <- M_i [indexer];

34: else

35: diff <- false;

36: SX[indexer] <- 0;

37: Mi[indexer] <- 0;

38: SX[indexer] = Mi[indexer];

39: Break loop

40: function Graph(){

41: node occ <- V

42: Seed <- V

43: function Seed<-V = TRUE is further augmented with the following:

44: if node V = singleton motif ε window

45: true;

ALGORITHM Maximal Motif Discovery for Biological Data in a Sliding Window

```
46:       else
47:                 false;
48:       End if
49:       End for
50:     }
51:        function search_tree(argument ℓ) {
52:        List<Integer> result = root.search(ℓ);
53:      if result == null
54:       Then
55:        Print ("Motif not found");
56:     else
57:       then
58:          ℓ Len = ℓ->length;
59:      for each i ε through results
60:      print "Motif found at position " and indexer - patLen;
61:          end
```

From lines 1 - 5 the user input has been defined. These inputs are as follows:

- A string ‘X’ consists of the complete genome sequence of the DNA.
- A Window Size ‘ℓ’ that splits the entire string into substrings.
- Mismatches occur in up to ‘d’ specific positions in the motif, known as ‘don’t care letters’
- ‘k’ sets a minimum threshold for occurrences of maximal motifs.

Line 6 defines the program's output, which is an array ‘SX’ of motifs, where SX[i] $=|M_{i,d,k}|$and $\ell \leq i < n$.

In line 7, I have passed the sequence of, https://www.ebi.ac.uk/ena/browser/api/fasta/EF445891.1?lineLimit=1000 as an argument to the method Suffix() that accepts a single argument of the type String array. This is also called java command line arguments.

In line 8, the constructor creates the root node, which is the first node that expands to its children on the left- and right-hand sides, adds a sentinel character to all the sequences and builds a trie of suffixes of the given sequences from real genome database, the program can read the entered genome sequence (in FASTA format) as an array. This sequence is taken as an input by the user in an array of string.

After reading the file (string X, the input of actual genome), an array of sequences is sent to the constructor to turn the input string X into its binary equivalent using the DNA alphabet's twobits-per-base encoding, as an example the string will divide into a series of substrings. (as shown in Table 4-3 and 4-4):

Table 4-3: String sequences

#	Sequence
1	ATGAAG$
2	ATCTTA$
3	ACTTTA$
4	ATTTTG$
5	ACCTTA$
6	ATCTTA$

Table 4-4: Binary encoding of string sequences

#	Sequence
1	001110000010
2	001101111100
3	000111111100
4	001111111110
5	000101111100
6	001101111100

In line 9, associated nodes are also created. The nodes are added to the vector of nodes throughout the process to distinguish different suffixes using queues. As a queue is created for every symbol present in the sequences, we only have to repeat the queues, and only one node has to be developed for each queue present. Those suffixes are substrings that begin at position 1 and always end with the last symbol of the sequence.

The Sliding Window method can then be applied to this complete set of strings that divides it into small blocks. The user specifies the window size if the user wishes to read the database partially. The total number of characters in the database is denoted by X. Let us suppose X=1000. If the user wishes to read-only 800 characters, then window size 1 will be 800.

The sentinel character (‘$’) is added to identify the suffix's end. This is called the flag value, which is the unique value that is used as a condition of termination, typically in a loop or recursive algorithm, lines (9-11) consider all suffixes of given string and insert them using recursive function insertSuffix() in SuffixTreeNode class. This value has a significant impact on our algorithm performance. It is used as one of the two termination conditions in our recursive construction algorithm, from line (12- 39) this termination can be in the case of either of the following scenarios:

- Mismatch: If the most significant number ‘*d*’ of allowed don’t care characters are reached.
- Match: If the smallest number ‘*k*’ for the rate of maximal motifs is reached.

In Line 12, a loop used check() method to check whether the specific set of characters are part of the given string or not. First, in (line 16) check if the array is empty or not then,

Lines (20 – 27) define Right_Handside function to find if given string is present with two or more match occurrence using find() method that searches the specified pattern or the expression in the given subsequence characterwise. First, adding a letter SX[i] = M_i to the right of the window, In line 22 the following cases are checked in order, if and only if $|occ(M_i)| \geq k$ in the window:

if M_i extends at least k occurrences of some motif $\widetilde{M} \in M_{i,d,k}$, it becomes the suffix of a new motif $\widetilde{M}'$. if the number of occurrences of $\widetilde{M}$ is equal to $\widetilde{M}'$, delet $\widetilde{M}$ from $M_{i,d,k}$, as it is no longer maximal.

Lines (28-39) define Left_Handside function to find if given string is present with one mismatch occurrence using diff() method that returns a boolean value true if the specified characters are substring of a given string and returns false otherwise. This, in turn, allows only one number of mismatches, d=1. Lines (31–39) the following cases are checked in order, several deletion conditions applied to update the motifs in the sliding window, which affecting the motif graph synchronously with the deletions, which are as follows:

if the key is unique, delete all the nodes.
SX [indexer] <- 0;
if the key having at least one other key as prefix key, delete nodes from the end until the first leaf node of the longest prefix key.
Starts loop from the end till the first node
M_i [indexer] <- 0;
SX [indexer] = M_i [indexer];
if the key is acting as prefix key for longer key, unmark leaf node.
M_i [indexer] <- null;
SX [indexer] <- M_i [indexer];

Maintaining the right- and left-most seeds in two areas checking their maximality (nodes) and modernizing their link with the rival seeds (edges) are the main technical challenges encountered during the process of reporting motifs. Both these processes work together to create a Sliding Window logic.

Therefore, mimicking the working of the sliding window. The sliding window strategy is used to evaluate the effectiveness of the motif discovery algorithm, and this technique does not require the creation of a structure.

Lines (40-50) define motif graph that holds the number of occurrences of V, (node occ <- V) in the window of ℓ-length on X, then check if node V is a seed, (Seed <- V) in the window, then return a boolean value true, and false otherwise.

The function "visualize()" creates a motif graph. This function calls another function "visualize_f(int, string)," which is responsible for the graph creation. Thus, the motif graph identifies the only subset of motifs that occur on both sides of the window and updates it competently.

For each node V current do (function Seed<-V = TRUE) is further augmented with the following:

if node V represents a singleton motif in the window, (if node V = singleton motif ε window), then return a boolean value true, which is Motif(V) and false otherwise.

Lines (51-61) Prints all occurrences of Motifs by calling a recursive search function for root of trie, will get a list of all indexes (where ℓ is present in text x) in variable 'result'. In (line 53) Check if the list of indexes is empty or not, if it's empty then print ("Motif not found"); Otherwise print the motifs found at all positions, the output is an array SX of motifs, where SX[i] $=|M_{i,d,k}|$and $\ell \leq i < n$.

The algorithm uses a trie data structure. The trie is saved into the memory in the form of a vector of nodes. The total construction time for this trie is equal to *k*. At every point, meaning after each sliding window, all the data regarding a single node is saved. The Boolean per child can be spent with the child having a size 1 rather than spending one father link per child. It is easy to move from adjacently placed siblings as the vectors have their constituents within the contiguous memory location. The index will be increased by 1 when the Boolean indicates that the sibling is not the last child when moving from one child to another. The process will be repeated until we reach the last child, which the Boolean will signal as the last one.

4.5 Experimental Results

The experiments of Maximal Motif Discovery in Sliding Window, MMDSW (Alshammary, Iliopoulos, Mohamed, & Vayani, 2020) were conducted in Java programming language on a Desktop PC using an Intel Pentium processor (CPU 4415U) with 2.30 GHz, 4.00 GB of RAM, 64-bit Windows 10, and the MS.Net Framework 4.5 platform was utilised for the proposed solution.

According to the experiments results, verified previous theoretical findings (Iliopoulos, Mohamed, Pissis, & Vayani, 2018) by implementing the algorithm (MMDSW) in and testing it with real genomic data, in order to discover all maximal motifs in a sliding window of ℓ-length on a sequence x of n-length. The purpose of implemented (MMDSW) algorithm is finding biologically significant regions in real genomic sequences.

The dataset used in this experiment is real genomic sequences (string x) from the European Nucleotide Archive (ENA). The ENA maintains extensive records of the world's nucleotide sequencing information, covering raw sequencing data, sequence assembly information, and functional annotation. In addition, a database of nucleotide sequences EF445891.1, Escherichia coli strain ATCC 11775 was also used [https://www.ebi.ac.uk/ena/data/view/EF445891]. The real genomic sequence readings for Escherichia coli are included in the dataset (As shown in Figure 4-4).

Figure 4-4 The Dataset of (MMDSW) Maximal Motif Discovery in a Sliding Window.

All the matches and mismatches, thresholds k<=2 and d=1 is tested to explain the window size, which is ℓ-length= 200. The sliding window moves in sequences 1-200, then 2-201, then 3-202, take into account all the n-ℓ windows of size ℓ provided the original string x has size n=3050 (As shown in Table 4-5).

The information of Maximal Motif Discovery in a Sliding Window (MMDSW) Application	
E. coli Sequence length n of string X	3050
Sliding window length ℓ	200
Note	It is important to note that the application is designed to allow the end user to provide values for all the inputs parameters (X, ℓ,K, d), and could accept any URL of any real genomic dataset. for testing the application in real data, we setup the parameters with the values shown in table in order to discover all maximal motifs in a sliding window.
	Also, note that the E. coli with no more than one mismatch d and the minimum number of occurrences k>=2 of maximal motifs.
E. coli complete sequence	ATGAATATCTTACATATATGTGTGACCTCAAAATGGTTTAATATTGACAACAAG ATTGTCGATCACCGCCCTTGATTTGCCCTTCTGTAGCCATCACCAGAGCCAAAC CGATTAGATTCAATGTGATCTATTTGTTTGCTATATCTTAATTTTGCCTTTTAC AAAGGCCATCTCTCGTTTATTTGCTTGTTTTAGTAAATGATGGTACTTGCATAT ATATCTGGCGAATTAATGAATATTGCAAATGTAATATTCACAGGGATCACTGTA ATTAAAATAAATGAAGGATTATGTAATGGAAAACTTTAAACATCTCCCTGAACC GTTCCGCATTCGTGTTATTGAGCCAGTAAAACGTACCACTCGCGCTTATCGTGA AGAGGCAATTATTAAATCCGGTATGAACCCGTTCTTGCTGGATAGCGAAGATGT GTTTATCGATTTACTGACCGACAGCGGCACCGGGGCGGTGACGCAGAGTATGCA GGCCGCGATGATGCGTGGCGACGAAGCCTACAGCGGCAGCCGCAGCTACTACGC GTTAGCCGAGTCAGTGAAAAATATCTTTGGTTATCAATACACTATTCCGACTCA CCAGGGCCGTGGTGCAGAGCAAATCTATATTCCGGTACTGATTAAAAAACGCGA GCAGGAAAAAGGCCTGGATCGCAGCAAAATGGTGGCGTTCTCTAACTATTTCTT TGATACCACGCAGGGCCATAGCCAGATTAACGGCTGTACCGTGCGTAACGTCTA TATCAAAGAAGCCTTCGATACTGGCGTGCGTTACGACTTTAAAGGCAACTTTGA CCTTGAGGGATTAGAACGCGGTATTGAAGAAGTTGGCCCGAATAACGTGCCGTA TATCGTTGCAACCATCACCAGTAACTCCGCAGGTGGTCAGCCGGTTTCACTGGC AAACTTAAAAGCGATGTACAGCATCGCGAAGAAATACGATATTCCGGTGGTCAT GGACTCCGCACGCTTTGCCGAAAACGCC

Table 4-5 The dataset information of (MMDSW) Maximal Motif Discovery in a Sliding Window.

It should be noted that the implemented bioinformatics application used can accept any URL of any real genomic dataset inputs for testing, to discover all maximal motifs in a sliding window of ℓ-length on a sequence x of n-length. The application Layout Panel (As shown in Figure 45):

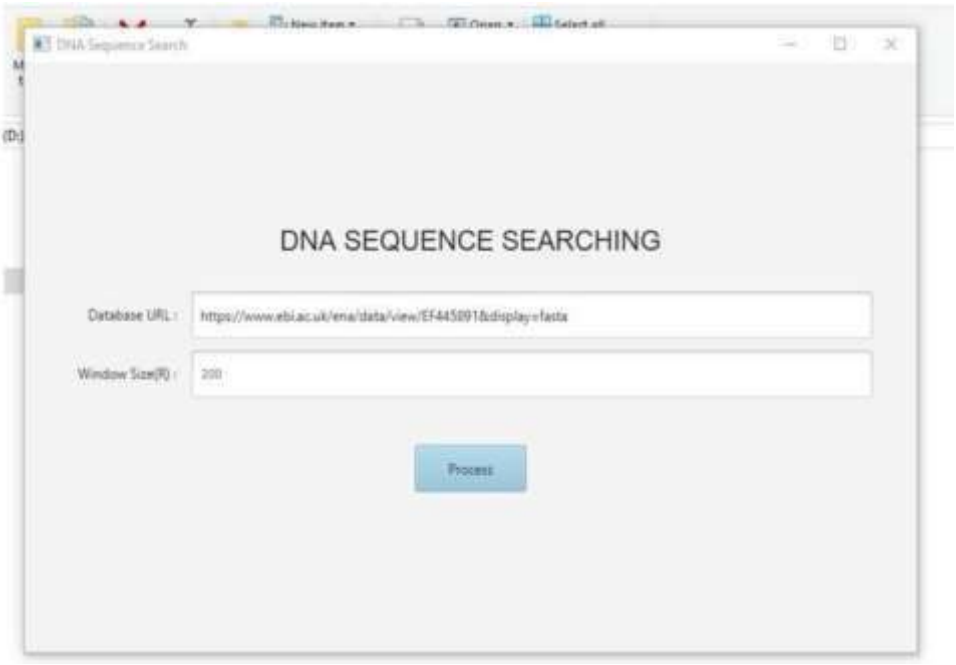

Figure 4-5 The (MMDSW) layout panel.

The application processes the input serially so that the entire input is not processed in a single step. We start by appending a string of ℓ letters not in Σ to the start of X. We also appended a unique letter not in Σ to the end of X to guarantee that the motif graph (augmented suffix tree ST) of the final window is explicit. The subset of motifs at the window end are identified and updated by the changes to the motif graph, by checking nodes maximality and updating their relationship with neighbouring seeds (edges). In "Suffix.java," the function "visualize()" creates a motif graph. This function calls another function "visualize_f(int, string)," which is responsible for the graph creation. The (MMDSW) output:

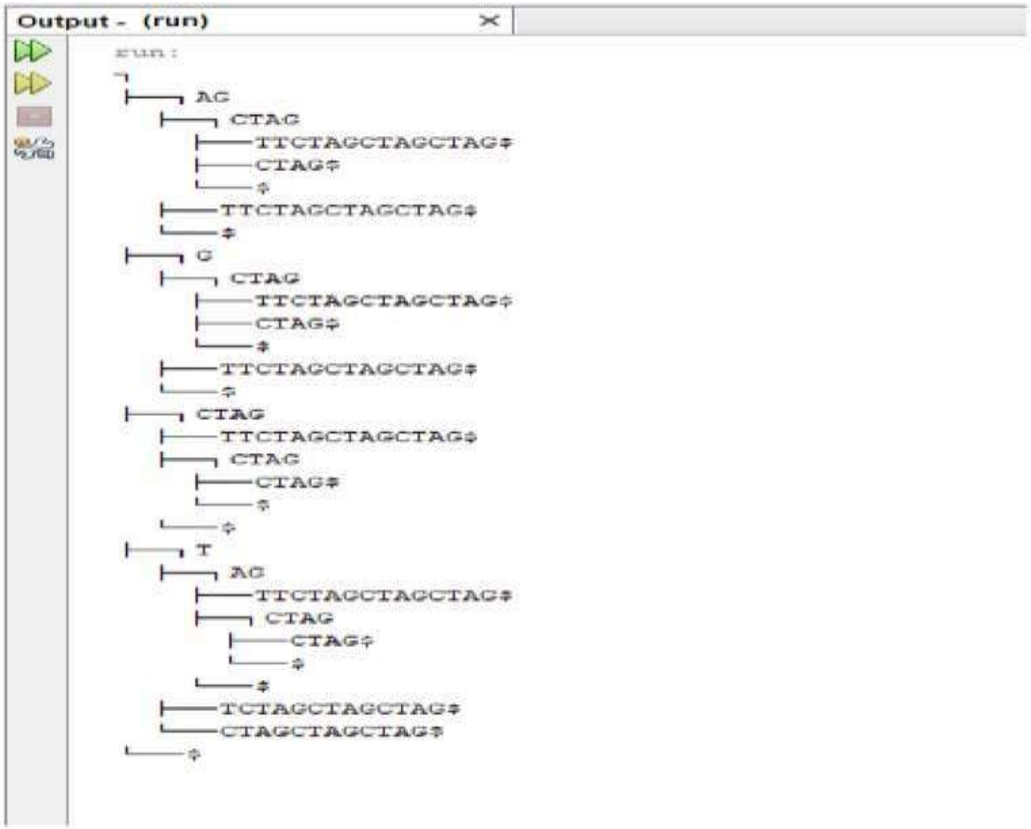

Figure 4-6 The Output of the (MMDSW) application.

The screenshot above is from the running program. It depicts the root node and child nodes as being split by the modifier $. The output is an array SX of motifs, where SX[i] =$|M_{i,d,k}|$ and $\ell \leq i < n$, for each occurrence of motif M_i under two conditions, K 'matches' and d 'mismatches' .

We designed a structure with a lower memory cost (16 bytes per character) to void the problems regarding memory and performance by minimizing pointers and fields. We represented our sequences using motif trie, which gave us a fast structure to find patterns by using two additional structures to keep the information of the suffixes and provide a fast way to search sequences IDs and suffixes in the trie—noting down the motif discovery algorithm, which considers the limitation regarding motif length, under two conditions *k* of the match, *d* of mismatch.

4.5.1 Time Complexity:

These findings show the potential to improve time and space to solve the crisis with the motif trie and the improvements related to the time required to solve the problem of maximal motif in sliding window algorithm using the motif trie. where, the time complexity is linear with O(n), looping n bits will produce the outcome only once as these loops through the BitMap only once. A character array is constructed by taking 8 bits together and transforming them into ASCII, which will happen along with the character occurring in only 1/64, where 1 byte equals 64 bits, this is a size of advanced processors called a machine word. This process will result in linear time complexity of O (n+n/64), the space complexity is O(ℓ2) in our algorithm.

The application's performance is assessed for memory and time (As shown in Figure 4-7):

```
Used memory is bytes: 304512
Used memory is megabytes: 0
Execution time in nanoseconds  : 15713800
Execution time in milliseconds : 15
BUILD SUCCESSFUL (total time: 0 seconds)
```

Runtime for the program: 15 ms or 0.015 seconds

Figure 4-7 The performance evaluation

The flowchart explains the complete program:

Maximal Motif Discovery in a Sliding Window (MMDSW)

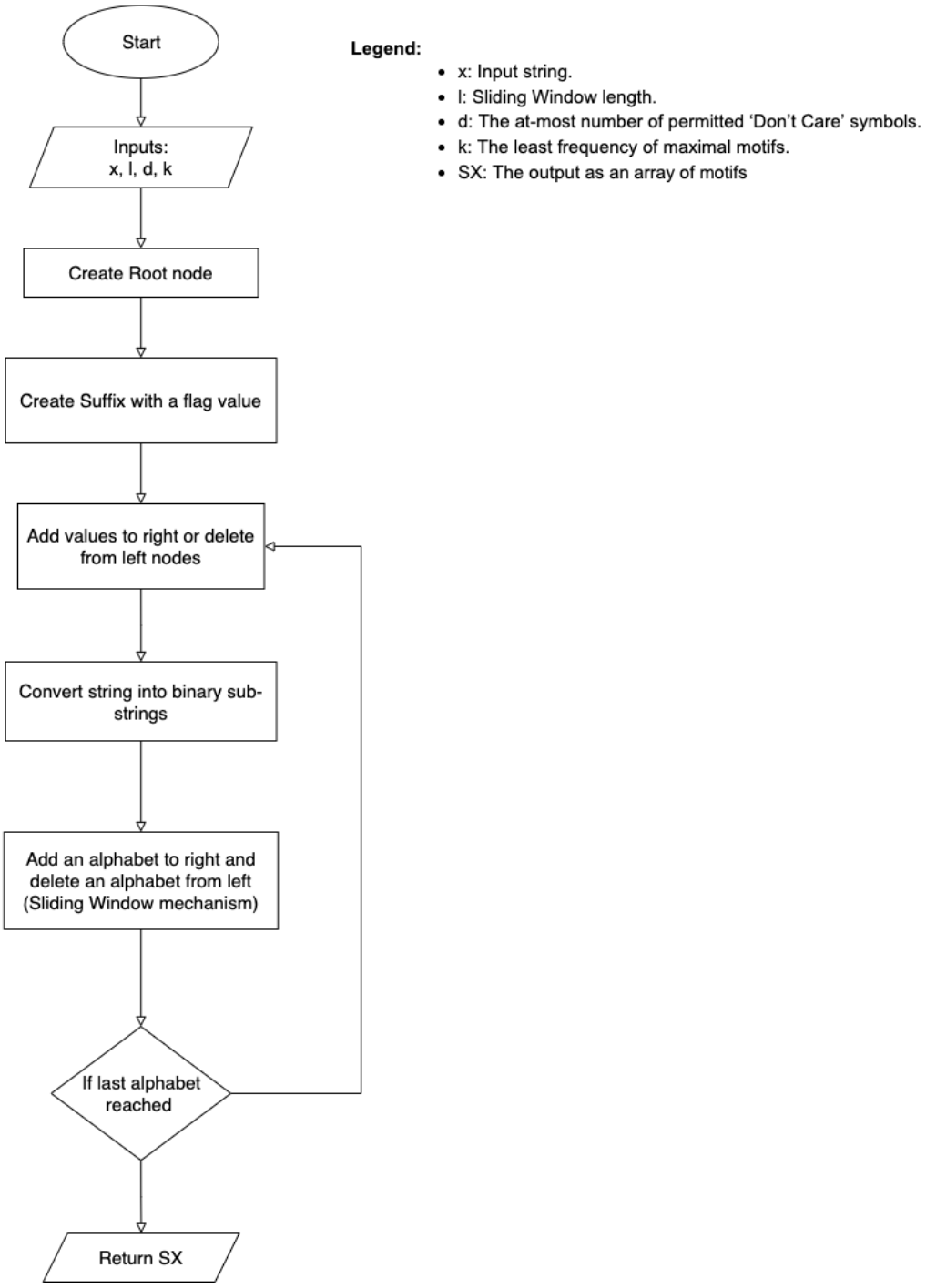

Figure 4-8 Flow chart of the proposed solution

Our application traits are elaborated in Table 4-6 given below:

• **Application**	• **Features**
• Maximal Motif Discovery in a Sliding Window (MMDSW) application	(1) Qualitative and quantitative (2) High sensitivity (capable of resolving complexes of different proteins or DNA by stoichiometry, even detecting conformational changes) (3) Particularly useful for analysing protein–DNA interactions in model bacterial species ranging from 120 to 300bp (for example, it is 240bp in E. coli) and for determining cell types that contain each binding activity

Table 4-6 The summary of the (MMDSW) application

4.6 Discussion and Future Work

Motif discovery is one of the sequence analysis problems under the application layer and it is one of the significant difficulties in bioinformatics applications. In this chapter, we implemented a maximal motif discovery in sliding window (MMDSW) algorithm to verified previous theoretical findings (Iliopoulos, Mohamed, Pissis, & Vayani, 2018) The proposed solution used the suffix tree, motif trie, to address the maximal motifs extraction problem from real genomic data with don't cares in a sequence x of n- length. The purpose of implemented the proposed algorithm is finding biologically significant regions by discover all maximal motifs $\widetilde{M}_{i,\,d,\,k}$ in a sliding window of ℓ-length on a sequence x of n-length.

In future work, this chapter's implementation of maximal motifs could be extended to flexible motifs. Refinement of these results could take place by replacing the don't care symbol with Kleene star symbol that matches any sequence, rather than a single character.

5 Concluding Observations

5.1 Thesis Summary

In genomic sequence analysis, this dissertation presented implementations and algorithms for problems relating to periods (also referred to as powers) and regularity in circular strings of repeating patterns. An overview of the dissertation's work is provided below, along with a discussion of the remaining problems and future research directions. Suffix trees are utilised extensively as computing tools in this dissertation.

In Chapter 2, we studied cyclic factors of strings. We proposed a several algorithms to computing k-cyclic periodicity of period ℓ of x, for a string x of length n and an integer k<n, where n is the length of the input string x, x=u1u2…uℓ, ui=c(uj), |ui|=|uj|=k, ∀ i,j, 1≤i≤ℓ, 1≤j≤ℓ, and k×ℓ=n.

In computational biology, where periods are linked to multiple regulatory systems and play a significant role in genomic sequence analysis, we developed algorithms for their efficient computation and implementations in a cyclic pattern for biological data in Chapter 3,4.

Our 'fast minutiae extraction algorithm' implementation for fingerprint matching was presented in Chapter 3. The proposed algorithm is based on circular string-matching technique that uses the information in string format converted by circle sequence method from the minutiae information. The purpose of creating the string of detailed information is to make it easy for implemented algorithm to compared it with records kept in database. The proposed algorithm aims to tackle the rotation problem in fingerprint recognition systems, increasing the performance and accuracy of the fingerprint identification system faster than the current solutions.

Our 'Maximal Motif Discovery in Sliding window algorithm, MMDSW' implementation for all maximal motifs in a sliding window of ℓ-length, on a sequence x of length n within terms of constraints in the search, where K is the minimum number of occurrences of maximal motifs $\widetilde{M}_{i,\,d,\,k}$ and d is the maximum allowed error(s), was presented in Chapter 4. In order to discover interesting regions by analyse real readings of genomic data from DNA patterns in molecular sequences.

5.2 Future Work

The possibility to extend the research work in the future as follows:

In Chapter 2, will be focused on computing the cyclic-periodic array, which is the cyclic periodicity of every prefix of string x. The cyclic periodicity will be extended to cover the case $u_1u_2u_3\ldots u_k\ u^1$, where $u_i=c(u_j)\ \forall$ i,j and u^1 is a substring of some u_i.

In Chapter 3, the algorithm will be expanded to include several different types of databases in recognition system besides fingerprint such as face print ...etc. to handle vast volumes of data, different recognition systems and biometric security difficulties. This can be achieved by extending the work in Chapter 2, on computing the cyclic-periodic array to match the query of several different types of databases in same recognition systemin an extensive database.

In Chapter 4, the algorithm of maximal motif discovery in sliding window will be expanded to include flexible motifs by replacing the don't care symbol with Kleene star symbol that matches any sequence, rather than a single character.

References

Ajala, et al. (2019). On the cyclic regularities of strings. IFIP Advances in Information and Communication Technology, 560, 219-224. https://doi.org/10.1007/978-3-030-19909-8_19

Ajala, O. I., Iliopoulos, C. S., & Khan, M. R. (2016). Identification of fingerprints using circular string approximation for mobile devices. Computer Science and Information Systems, 13, 193197, do: 10.15439/2017F3

Ajala, O., Aljamea, M., Alzamel, M., Iliopoulos, C. S., Strigini, Y. (2016). Fast fingerprint recognition using circular string pattern matching techniques. Patterns 2016: The Eighth International Conferences on Pervasive Patterns and Applications.

Alatabbi, A. (2014). Advances in stringology and applications. PhD thesis, Natural and Mathematical Sciences, King's College London.

Alatabbi, A., Al-Jamea, M., & Iliopoulos, C. S. (2013). Malware detection using computational biology tools. International Journal of Engineering and Technology 5(2), 315.

Allers, T., & Mevarech, M. (2005). Archaeal genetics – the third way. Nat Rev Genet, 6 58-73. doi: 10.1038/nrg1504.

Alshammary M.H., Iliopoulos C.S., & Khan M.R. (2020). Fingerprints recognition systembased on mobile device identification using circular string pattern matching techniques. In: Maglogiannis I., Iliadis L., & Pimenidis E. (eds) Artificial Intelligence Applications and Innovations. AIAI 2020 IFIP WG 12.5 International Workshops. AIAI 2020. IFIP Advances in Information and Communication Technology, vol 585. Springer, Cham. https://doi.org/10.1007/978-3-030-49190-1_20.

Alshammary M.H., Iliopoulos C.S., Mohamed M., & Vayani F. (2020) Application and algorithm: Maximal motif discovery for biological data in a sliding window. In: Maglogiannis I., Iliadis L., Pimenidis E. (eds) Artificial Intelligence Applications and Innovations. AIAI 2020
IFIP WG 12.5 International Workshops. AIAI 2020. IFIP Advances in Information and Communication Technology, vol 585. Springer, Cham. https://doi.org/10.1007/978-3-03049190-1_19.

Alshammary, M. H., & Alanezi, F. A. (2017). A Review of Recent Developments in NOMA & SCMA Schemes for 5G Technology, 2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES), Anyang, 2017, pp. 55-59, doi: 10.1109/DCABES.2017.19. https://ieeexplore.ieee.org/document/8253035

Antoniou, P., Daykin, J., Iliopoulos, C., Kourie, D., Mouchard, L., & Pissis, S. (2009). Mapping uniquely occurring short sequences derived from high throughput technologies to a reference genome. In Information Technology and Applications in Biomedicine, 2009. ITAB 2009. 9th International Conference, IEEE, pp. 1–4.

Arimura, H., & Uno, T. (2007). An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. Journal of Combinatorial Optimization, 13(3), 243-262.

Bailey, T.L., & Elkan, C. (1994). Fitting a mixture model by expectation maximization to discover motifs in biopolymers.

Barton, C., Iliopoulos, C. S., & Pissis, S. P. (2014). Fast algorithms for approximate circular string matching. Algorithms for Molecular Biology 9, 9.

Baxevanis, A. D., Bader, G. D., & Wishart, D. S. (2020, eds.). Bioinformatics, vol. 372, John Wiley & Sons, 2020, pp. 793-795.

Bioinformatics Organization. Bioinformatics. [Online] Available from: https://www. bioinformatics.org/. [Last accessed: 10.2016].

Buhler, J., & Tompa, M. (2002). Finding motifs using random projections. Journal of Computational Biology, 9(2), 225–242.

Carlson, J. M., Chakravarty, A., & Gross, R. H. (2006a). Beam: a beam search algorithm for the identification of cis-regulatory elements in groups of genes. Journal of Computational Biology, 13(3), 686–701.

Carlson, J. M., Chakravarty, A., Khetani, R. S., & Gross, R. H. (2006b). Bounded search for de novo identification of degenerate cis-regulatory elements. BMC Bioinformatics, 7(1), 254.

Carvalho, A. M., Freitas, A. T., Oliveira, A. L., & Sagot, M.-F. (2004). Efficient extraction of structured motifs using box-links. In International Symposium on String Processing and Information Retrieval, pp. 267–268. Springer.

Carvalho, A. M., Freitas, A. T., Oliveira, A. L., & Sagot, M.-F. (2006b). An efficient algorithm for the identification of structured motifs in dna promoter sequences. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), 3(2), 126–140.

Christian, C., & Lecroq, T. (2004). Handbook of exact string matching algorithms. London, UK: Kings College Publications.

Clifford, R., & Iliopoulos, C. (2004). Approximate string matching for music analysis. Soft Computing 8, 9, 597–603.

Communications Security Establishment. (2020).The security assessment and authorization (SA&A) [Online] Available from: www.csecst.gc.ca process for National Security Systems

Crochemore, M., Hancart, C., & Lecroq, T. (2007). Algorithms on strings. Cambridge University Press, New York, NY, USA.

Defant, C. (2016). Anti-power prefixes of the thue-morse word. arXiv preprint, arXiv:1607.05825.

Dulbecco, R., & Vogt, M. (1963). Evidence for a ring structure of polyoma virus DNA. In: Proc Natl Acad Sci 50(2), 236–243, doi: 10.1073/pnas.50.2.236.

Erdős, P. et al. (1973). Anti-ramsey theorems

Eskin, E. & Pevzner, P. A. (2002). Finding composite regulatory patterns in dna sequences. Bioinformatics, 18(suppl_1), S354–S363.

Farach, Martin (1997), "Optimal Suffix Tree Construction with Large Alphabets", 38th IEEE Symposium on Foundations of Computer Science (FOCS '97), pp. 137–143.

Fernandes, F., Pereira, L., & Freitas, A. (2009). CSA: An efficient algorithm to improve circular DNA multiple alignment. BMC Bioinformatics 10, 1–13. doi: 10.1186/1471-2105-10-1.

Fiala, E.R., & Greene, D.H. (1989). Data compression with finite windows. Commun. ACM, 32, 490–505.

Frith, M. C., Saunders, N. F., Kobe, B., & Bailey, T. L. (2008). Discovering sequence motifs with arbitrary insertions and deletions. PLoS computational biology, 4(5), e1000071.

Fujita, S., Magnant, C., & Ozeki, K. (2010). Rainbow generalizations of ramsey theory: a survey. Graphs and Combinatorics 26(1), 1–30.

Fuller, R.S., Funnell, B. E., & Kornberg, A. (1984). The dnaA protein complex with the E. coli chromosomal replication origin (oriC) armcl other DNA sites. Cell 38(3), 889- 900.

GenBank and W. Statistics. [Online] Available from: https://www.ncbi.nlm.nih.gov/genbank/statistics/, 2020.

Grossi, R., & Italiano, G. F. (1993). Suffix Trees and their Applications in String Algorithms

Grossi, R., Menconi, G., Pisanti, N., Trani, R., & Vind, S. (2018). Motif trie: An efficient text index for pattern discovery with don't cares. Theoretical Computer Science, 710, 74-78.

Grossi, R., Pietracaprina, A., Pisanti, N., Pucci, G., Upfal, E., & Vandin, F. (2011). Madmx: A strategy for maximal dense motif extraction. Journal of Computational Biology, 18(4), 535– 545.

Gusfield, D. (1997). Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology. Cambridge University Press.

Güting, R.H., Behr, T., & Nidzwetzki, J. K. (2021). Distributed arrays: an algebra for generic distributed query processing. Distributed and Parallel Databases, 1-56.

Henikoff, S., Henikoff, J. G., Alford, W. J., & Pietrokovski, S. (1995). Automated construction and graphical presentation of protein blocks from unaligned sequences. Gene, 163(2), GC17– GC26.

Henry, E. R. (1900). Classification and Uses of Finger Prints. Routledge.

Hertz, G. Z. & Stormo, G. D. (1999). Identifying dna and protein patterns with statistically significant alignments of multiple sequences. Bioinformatics (Oxford, England), 15(7), pp. 563–577.

ICANN. List of top-level domains. [Online] Available from: https://www.icann.org/resources/pages/ tlds-2012-02-25-en. [Last accessed: 11.2015].

Iliopoulos, C.S., Mohamed, M., Pissis, S.P., & Vayani, S.F. (2018). Maximal motif discovery in a. sliding window- In: SPIRE. vol. 11147, pp. 191-205, 2018.

Waterman, M.S., General methods of sequence comparison. Bulletin of Mathemat— ical Biology 46(4), 47.W.500, 1984.

IUPAC-IUB Commission on Biochemical Nomenclature (1971). Abbreviations and symbols for nucleic acids, polynucleotides and their constituents. Archives of Biochemistry and Biophysics.

Jain, A. K., Prabhakar, S., Hong, L., & Pankanti, S. (2000). Filterbankbased fingerprint matching. Image Processing, IEEE Transactions vol 9, 5, 846–859.

Jonassen, I., Collins, J. F., & Higgins, D. G. (1995). Finding flexible patterns in unaligned protein sequences. Protein Science, 4(8), 1587–1595.

Jones, N. C. & Pevzner, P. A. An introduction to bioinformatics algorithms.

Kai, C., Xin, Y., Xinjian, C., Yali, Z., Jimin, L., & Jie, T. (2012). A novel ant colony optimization algorithm for large-distorted fingerprint matching, Pattern Recognition, 45(1), 151-161.

Keich, U. & Pevzner, P.A. (2002). Finding motifs in the twilight zone. In Proceedings of the sixth annual international conference on Computational biology, pp. 195–204. ACM.

Kiesel et al. (2018). The bamm web server for de-novo motif discovery and regulatory sequence analysis. Nucleic acids research.

Kolpakov, R., Bana, G., & Kucherov, G. (2003). mreps: efficient and flexible detection of tandem repeats in dna. Nucleic Acids Research, 31(13), 3672–3678.

Korean Information Society Agency (2011). Development of Korean Smartphone Addiction Proneness Scale for Youth and Adults. Seoul: Korean Information Society Agency.

Landau, G. M., & Vishkin, U. (1998). Fast string matching with k differences. Journal Of Computer and System Sciences, 37, 63-78, 1998.

Larsson, N. J. (1996). Extended application of suffix trees to data compression. In: IEEE Data Compression Conf., 180-189.

Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F., & Wootton, J. C. (1993). Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment.
science, 262(5131), 208–214.

Lee, T. et al. (2010). Finding optimal alignment and consensus of circular strings. In: Proceedings of the 21st annual Conference on Combinatorial Pattern Matching (2010), 310– 322.

Leonard, A. C. and Méchali, M. (2013). Dna replication origins. Cold Spring Harbor perspectives in biology, 5(10):a010116.

Leung, H. C., & Chin, F. Y. (2006). An efficient motif discovery algorithm with unknown motif length and number of binding sites. International Journal of Data Mining and Bioinformatics, 1(2), 201–215.

Lin, H., Jain, A., Pankanti, S., & Bolle, R. (1997). Identity authentication using fingerprints. In Audio and Video based Biometric Person Authentication, Springer.

Lipps, G. (2008). In: Plasmids: Current Research and Future Trends. Expert Reviews

Luscombe, N. M., Greenbaum, D., & Gerstein, M. Whatisbioin Formatics: an introduction and overview. Yearbook of Medical Informatics 1 (2001), 83–99.

Maltoni, D., Maio, D., Jain, A.K., & Prabhakar, S. (2009). Handbook of fingerprint recognition. Springer-Verlag.

Marsan, L., & Sagot, M., F. (2000). Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. Journal of Computational Biology, 7, 345-360.

McCreight, E.M. (1976). A space-economical suffix tree construction algorithm. Journal of the ACM (JACM) 23(2), 262–272.

Meijer, M. et al. (1979). Nucleotide sequence of the origin of replication of the Escherichia coli K-12 chromosome. Proceedings of the Nationål Academy of Sciences, 76(2), 580-584.

Merriam-Webster, (2016). Algorithm, [ONLINE] Available at: http://www.merriamwebster.com/dictionary/algorithm , 2016.

Modan K Das, & Ho-Kwok Dai (2007). A survey of DNA motif finding algorithms, Volume 8 of BMC Bioinformatics (2007), pp. S21–S21.

Mohapatra, C., & Pandey, M. (2015). A review on current methods and application of digital image steganography. International Journal of Multidisciplinary Approach & Studies 2, 2.

Mosig, A. et al. (2006). Comparative analysis of cyclic sequences: viroids and other small circular RNAs. In: German Conference on Bioinformatics, Volume 83 of LNI (2006), pp. 93– 102.

Narayanan, S. (2017). Functions on antipower prefix lengths of the thue-morse word.

Nature Education. DNA sequencing. [Online] Available from: http://www.nature.com/scitable/ content/dna-sequencing-6656663, 2016.

Nguyen, T., Wang, Y., & Li, R. (2013). An improved ridge features extraction algorithm for distorted fingerprints matching. J. Inf. Secur. Appl., 18, 206–214.

Parida, L., Rigoutsos, I., & Platt, D. (2001). An output-sensitive flexible pattern discovery algorithm. In Annual Symposium on Combinatorial Pattern Matching, pp. 131–142. Springer.

Pavesi, G., Mercghetti, P., Mauri, G., & Pesole, G. (2004). Weeder web: Discovery of transcription factor binding sites in a set of sequences from coregulated genes. Nucleic Acids Research 32(Web-Server-Issue), 199-203.

Pevzner, P. A., & Sze, S.-H. (2000). Combinatorial approaches to finding subtle signals in DNA sequences. ISMB, 8, 269–278.

Pisanti, N., Carvalho, A. M., Marsan, L., & Sagot, M.-F. (2006b). Risotto: Fast extraction of motifs with mismatches. In Latin American Symposium on Theoretical Informatics, pp. 757– 768. Springer.

Pissis, S. (2016). Lecturenotes in algorithms for computational molecular biology.

Pissis, S. P. (2014). MoTeX-ll; structured MoTif eXtraction from large-scale datasets. BMC Bioinformatics, 15, 235.

Pissis, S. P., Stamatakis, A., Pavlidis, P. (2013). MOTeX: A word-based HPC tool for rnotif extraction. In Gao, J. (ed.) ACM Conference on Bioinformatics, Computational Biology and Biomedical Informatics. ACM-BCB 2013, Washington, DC, USA, September 22-25, p. 13.

Price, A., Ramabhadran, S., & Pevzner, P. A. (2003). Finding subtle motifs by branching from sample strings. Bioinformatics, 19(suppl_2), ii149–ii155.

Prober, J. M. et al. (1987). A system for rapid DNA sequencing with fluorescent chainterminating dideoxynucleotides. Science 238, 4825, 336–341.

Zhang, Q., Huang, K.: Hong Yan (2001). Fingerprint classification based on extraction and analysis of singularities and pseudoridges. In Fingerprint Classification Based on Extraction and Analysis of Singularities and Pseudoridges (Sydney, Australia, 2001), VIP 2001.

Rigoutsos, I. & Floratos, A. (1998). Motif discovery without alignment or enumeration. In Proceedings of the second annual international conference on Computational molecular biology, pp. 221–227. ACM.

Rigoutsos, I., Floratos, A., Ouzounis, C., Gao, Y., & Parida, L. (1999). Dictionary building via unsupervised hierarchical motif discovery in the sequence space of natural proteins. Proteins: Structure, Function, and Bioinformatics, 37(2), 264–277.

Sagot, M.-F. (1998). Spelling approximate repeated or common motifs using a suffix tree. In Latin American Symposium on Theoretical Informatics, pp. 374–390. Springer.

Sandve, G. K. & Drabløs, F. (2006). A survey of motif discovery methods in an integrated framework. Biology Direct, 1(1), 11.

Schwartz, D. & Gygi, S. P. (2005). An iterative statistical approach to the identification of protein phosphorylation motifs from large-scale data sets. Nature Biotechnology, 23(11), 1391.

Sebastian, S. (2013). Literature survey on automated person identification techniques. International Journal of Computer Science and Mobile Computing, 2(5), 232-237.

Senft, M. (2005). Suffix tree for a sliding window: An overview. WDS, 5, 41–46.

Shaffer, C. A. (2011). Data Structures & Algorithm Analysis in Java. Courier Corporation.

Sinha, Sr, & Tompa, M. (2003). YMF: a program for discovery of novel transcription factor binding sites by statistical overrepresentation. Nucleic Acids Research 31(13), 3586-3588.

Skiena, S. S. (2008). The algorithm design manual, vol. 2. Springer Science & Business Media.

Sruthy S. (2013). Literature survey on automated person identification techniques. International Journal of Computer Science and Mobile Computing, 2(5), 232–237.

Szor, P. (2005). The art of computer virus research and defense. Addison-Wesley Professional.

Tan, X. & Bhanu, B. (2002). Robust fingerprint identification. In International Conference on Image Processing 2002, vol. 1, IEEE, pp. I– 277.

Tan, X. & Bhanu, B. (2006). Fingerprint matching by genetic algorithms. Pattern Recognition 39(3), 465–477.

Thanbichler, M., Wang, S. C., & Shapiro. L. (2005). The bacterial nucleoid: A highly organized and dynamic structure. J Cell Biochem, 96(3), 506–521. doi: 10.1002/jcb.20519.

The Office for National Statistics. [Online] Available from: https://www.ons.gov.uk/, 2016.

The Open Web Application Security Project (OWASP) [Online] Available from: https://owasp.org/, 2020.

Thue, A. (1906). Uber unendliche zeichenreihen. Norske Vid Selsk. Skr. I Mat-Nat Kl.(Christiana) 7, 1–22.

Ukkonen, E. (1992). Constructing suffix trees on-line in linear time. In: Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture Information Processing '92, Volume 1-Volume I, pp. 484–492. North-Holland Publishing Co.

Ukkonen, E. (1995). On-line construction of suffix trees. Algorithmica, 14(3), 249–260.

Ukkonen, E. (1992). Approximatestring-matchingwithq- gramsandmaximalmatches. Theor Comput Sci, 92(1), 191–211.

Unar, J., Seng, W. C., & Abbasi, A. (2014). A review of biometric technology along with trends and prospects. Pattern Recognition, 47(8), 2673-2688.

van Helden, J., Andre, B., & Collado-Vides, J. (1998). Extracting regulatory sites from the upstream region Of yeast genes by computational analysis of oligonueleotide fre— quencies, Journal of Molecular Biology 281(5), 827-842.

Vanet, A., Marsan, L.,& Sagot, M.-F. (1999). Promoter sequences and algorithmical methods for identifying them. Research in Microbiology, 150(9-10), 779–799.

Vanet, A., Marsan, L., Labigne, A., & Sagot, M.-F. (2000). Inferring regulatory elements from a whole genome. an analysis of helicobacter pyloriσ80 family of promoter signals1. Journal of Molecular Biology, 297(2), 335–353.

Watson, J. D., & Crick, F. H. (1953). Molecular structure of nucleic acids. Nature, 171(4356), 737–738.

Weil, R. & Vinograd, J. (1963). The cyclic helix and cyclic coil forms of polyoma viral DNA. Proc Natl Acad Sci 50(4), 730–738. doi: 10.1073/pnas. 50.4.730.

Weiner, P. (1973). Linear pattern matching algorithms. In: 14th Annual Symposium on Switching and Automata Theory (swat 1973), pp. 1–11. IEEE.

Williams, R. (2014). Cyber crime costs global economy $445 bn annually. [Online] Available from: http://www.telegraph.co.uk/technology/internet-security/10886640/Cybercrime-costsglobal-economy-445-bn-annually.html, 2014.

Wurpel, D.J., Beatson, S.A., Totsika, M., Petty, N.K., & Schembri, M.A. (2013). Chaperoneusher fimbriae of escherichia coli. PloS One 8(1), e52835.

Zare-Mirakabad, F., Davoodi, P., Ahrabian, H., Nowzari-Dalini, A., Sadeghi, M., & Goliaei, B. (2009). Finding Motifs based on Suffix Trie, Advanced Modeling and Optimization, 177-194.

Appendix

1. Fingerprints Implementation Code:

```
using System;
using System.Collections.Generic;
using System.Linq; using System.Text;
using System.Threading.Tasks;

namespace MyUtils.MyHelpers
{
  public class StringUtils
    {
        public class FragmentItem
        {
            public int StartIndex { get; set; }
public int Length { get; set; }
  public string Fragment { get; set; }
             public FragmentItem() {
}
             public FragmentItem(int startIndex, int length, string
fragment)              {
               StartIndex = startIndex;
               Length = length;
               Fragment = fragment;
            }
        }

        /// <summary>
        /// Breaks the string str into f fragments.
Useful for circular string matching, asmf, etc.
        /// </summary>
  /// <returns></returns>
  public static List<FragmentItem> GetFragments(int f, string str)
{
            List<FragmentItem> result = new List<FragmentItem>();
int m = str.Length;
    int modulo = m % f;
   double nf = (double)m / f;

 //string.Format("f={0}, m={1}, modulo={2}, nf={3}", f, m, modulo,
nf).Dump();
   int first, len;
            //            char[] stArr = str.ToArray();

 for (int i = 0; i < f; i++)
            {
if (i < modulo)
                {
    first = i * (int)(Math.Ceiling(nf));//first = i * (ceil(nf));
     //last = first + (int)Math.Ceiling(nf) - 1;//last = first + ceil(nf)
-      1;
                    len = (int)Math.Ceiling(nf);
```

```
                }
  else                  {
    first = i * (int)(Math.Floor(nf)) + modulo;//first = i * (floor(nf)) +
modulo;
                    //last = first + (int)Math.Floor(nf) - 1;//last = first + floor(nf)
-      1;
                    len = (int)Math.Floor(nf);
                }
                result.Add(new FragmentItem(first,len, str.Substring(first, len)));
//string.Format("i={0}, first={1}, last={2}, frag={3}", i, first, last,
str.Substring(first, len)).Dump();
            }
              return
result;          }
}
    public static class StringExtensions
    {
        public static string x2m1(this string str)
        {
            if ((!string.IsNullOrWhiteSpace(str)) && (str.Length > 0))
            {
                return str + str.Substring(0, str.Length - 1);
            }             else
{
                return null;
            }
        }
   public static List<string> AllRotationsOfx2m1(this string cStr)
  {
            List<string> result = new List<string>();

  string x2m1 = cStr.x2m1();

            result = x2m1.AllRotations();
return result;
    }

 public static int StartIndexByLexOrder(this string str)
        {
            char[] cArr = str.ToArray();

            List<int> indices = new List<int>();
            indices.AddRange(Enumerable.Range(0, str.Length));
            //              string.Join(" ", indices).Dump();
            //              string.Join(" ", cArr).Dump();
              while (indices.Count() >
1)
            {
                List<int> tmp = new List<int>();
                for (int idx = 0; idx < indices.Count() - 1; idx += 2)
                {                     if (cArr[indices.ElementAt(idx)] <
cArr[indices.ElementAt(idx + 1)])
                    {
                        tmp.Add(indices.ElementAt(idx));
                    }
```

```
      else if (cArr[indices.ElementAt(idx)] > cArr[indices.ElementAt(idx + 1)])
                    {
                        tmp.Add(indices.ElementAt(idx + 1));
                    }                    else
{
                        int sprint1 = 0;
                 int sprint2 = 0;
while (((indices.ElementAt(idx) + sprint1) < cArr.Length) && (cArr[indices.ElementAt(idx)
+ sprint1] != '1')) sprint1++;
// can be further generalized
 while    (((indices.ElementAt(idx    +    1)    +    sprint2)    <    cArr.Length)    &&
(cArr[indices.ElementAt(idx +
1) + sprint2] != '1')) sprint2++; // can be further generalized
if (sprint2 > sprint1)
                        {
                            tmp.Add(indices.ElementAt(idx + 1));
                        }                          else
{
tmp.Add(indices.ElementAt(idx));
                        }
                    }
}

      if (indices.Count() % 2 ==
1)
                {
                    tmp.Add(indices.ElementAt(indices.Count() - 1));                 }
                  indices =
tmp;
                //          indices.Dump();
            }
              return
indices.First();
        }
        public static string Rotation(this string str, int r)
        {
            r = r % str.Length;
            if (!string.IsNullOrWhiteSpace(str))
            {
                return str.Substring(r) + str.Substring(0, r);
            }             else
{
                return str;
            }
}
```

```
  public static List<string> AllRotations(this string cStr)
```

```
        {
            List<string> result = new List<string>();

            StringBuilder cStrR = new StringBuilder(cStr);
    for (int i = 0; i < cStr.Length; i++)
            {
result.Add(cStrR.ToString());

                // prepare the next rotation of the string
                char fCh = cStrR[0];// if cStrR is ABC, then fCh is A now
cStrR.Remove(0, 1);// cStrR is BC now
cStrR.Append(fCh.ToString()); // cStrR is BCA now             }
             return
result;
        }
   public static string CharwiseCompare(this string str1, string str2)
        {
            if
((!string.IsNullOrWhiteSpace(str1))              &&
(!string.IsNullOrWhiteSpace(str2)))
            {
                StringBuilder sb = new StringBuilder();
char[] cArr1 = str1.ToArray();
       char[] cArr2 = str2.ToArray();
              if (cArr1.Length < cArr2.Length)
                {
         int i = 0;
          for (i = 0; i < cArr1.Length; i++)
                    {
                        if (cArr1[i] == cArr2[i])
                        {
sb.Append(cArr1[i]);                          }
else                           {
sb.Append("-");                             }
}                      for (; i < cArr2.Length;
i++)
                    {
sb.Append("-");                       }
}                  else
{                      int i =
0;                      for (i = 0; i < cArr2.Length;
i++)
                    {
                        if (cArr1[i] == cArr2[i])
                        {
                            sb.Append(cArr1[i]);
                        }
                        else
{                               sb.Append("-
");                           }
}                      for (; i < cArr1.Length;
i++)
                    {

                     sb.Append("-");
```

```
                    }
}
                return sb.ToString();
            }
else             {
return null;
            }
}
        public static string AngularRotation(this string str, double angleInDegree)
        {
            if (!string.IsNullOrWhiteSpace(str))
            {
 int rotIndex = str.RotationAngle2Index(angleInDegree);
                 return
str.Rotation(rotIndex);
            }
else             {
return null;
            }
        }

 public static double RotationIndex2Angle(this string str, int RotationIndex)
        {
            double result = 0.0;
  if ((!string.IsNullOrWhiteSpace(str)) && (str.Length > 0))
            {
                return ((double)(RotationIndex % str.Length) / str.Length) * 360;
}
              return
result;
        }
 public static int RotationAngle2Index(this string str, double RotationAngle)
{
            int result = 0;
    if (!string.IsNullOrWhiteSpace(str))
            {
                result = (int)Math.Ceiling((RotationAngle * str.Length) / 360);
            }
return result;
        }
    } }
```

2. Maximal Motif Discovery in a Sliding Window (MMDSW) Implementation code:

2.1 Node.java code:

```
package Business_Objects;

import java.util.ArrayList;
import java.util.List;

public class Node
{
   String sub = "";
   List<Integer> ch = new ArrayList<>();

}
```

2.2 Window.java code:

```
package Business_Objects;

public class Window {

   public void MMDSW(String X,int windowSize,int minThreshold,int mismatchOccur) {


   }
}
```

2.3 Suffix.java code:

```
package Business_Objects;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.List; import
java.lang.*; public class Suffix
{
   private List<Node> nodes = new ArrayList();
public Suffix(String str) {
this.nodes.add(new Node());
  for(int i = 0; i < str.length(); ++i)
{
   this.addSuffix(str.substring(i));
      }
   }

 public static String fromUnicode(String unicode) {
       String str = unicode.replace("\\", "");
      String [] arr = str.split("u");
       StringBuffer text = new StringBuffer();
 for (int i = 1; i < arr.length; i++) {
     int hexVal = Integer.parseInt(arr[i],16);
text.append(Character.toChars(hexVal));
       }
       return text.toString();
   }
   public static String toUnicode(String text) {
StringBuffer sb = new StringBuffer();
  for (int i = 0; i < text.length(); i++) {
  int codePoint = text.codePointAt(i);
       if
(codePoint > 0xffff) {
      i++;
         }
         String hex = Integer.toHexString(codePoint);
sb.append("\\u");
```

```
   for (int j = 0; j < 4 - hex.length(); j++) {
sb.append("0");
          }
          sb.append(hex);
       }
       return sb.toString();
   }
   private void addSuffix(String suf) {
int n = 0;        int n2;        label37:
for(int i = 0; i < suf.length(); n = n2) {
char b = suf.charAt(i);
  List<Integer> children = ((Node)this.nodes.get(n)).ch;
for(int x2 = 0; x2 != children.size(); ++x2)
 {
n2 = (Integer)children.get(x2);
 if
(((Node)this.nodes.get(n2)).sub.charAt(0) == b)
{ String sub2 = ((Node)this.nodes.get(n2)).sub;
               int j;

                  for(j = 0; j < sub2.length(); ++j) {
   if (suf.charAt(i + j) != sub2.charAt(j)) {
int n3 = n2;
    n2 = this.nodes.size();
  Node temp = new Node();
   temp.sub = sub2.substring(0, j);
 temp.ch.add(n3);
this.nodes.add(temp);

 ((Node)this.nodes.get(n3)).sub = sub2.substring(j);
((Node)this.nodes.get(n)).ch.set(x2, n2);
        break;
                    }
    }


  i += j;
```

```
 continue
label37;
            }
          }
          n2 = this.nodes.size();
Node temp = new Node();
temp.sub = suf.substring(i);
this.nodes.add(temp);
children.add(n2);
   return;
      }
   }
   public void visualize() {
  if (this.nodes.isEmpty()) {
          System.out.println("<empty>");
      } else {
this.visualize_f(0, "");
      }
   }


   private void visualize_f(int n, String pre) {
   List<Integer> children = ((Node)this.nodes.get(n)).ch;
 PrintStream var10000;
Object var10001;
   if (children.isEmpty()) {
var10000 = System.out;
var10001 = this.nodes.get(n);
          System.out.println("\u002d" + ((Node)var10001).sub);
      }
     else {
          var10000 = System.out;
  var10001 = this.nodes.get(n);
          System.out.println( fromUnicode("\\u2510") + ((Node)var10001).sub);
 for(int i = 0; i < children.size() - 1; ++i) {
   Integer c = (Integer)children.get(i);
              System.out.print(pre + fromUnicode("\\u251c\\u2500"));
```

```
 this.visualize_f(c, pre + " ");
        }
        System.out.print(pre + fromUnicode("\\u2514\\u2500"));
this.visualize_f((Integer)children.get(children.size() - 1), pre + "  ");
      }
   } }
```

2.4 left_handside.java code:

```
package Business_Objects; public class
Left_HandSide {      public void leftHandSide(int
score,Node ST,int j) {

   }
   public void copySeedLHS(int score,Node V,Node U) {

   }
   public void deleteRelevantEdge(int score,int edge) {

   }
   public void checkPathsFromSeedSuffix(int score,Node V,Node Z) {

   }
   public void deleteEdge(int score,int edge) {

   }
   public void addEdge(int score,Node s) {

   }
   public void checkRelevantEdge(int score,Node V,Node Z) {

   }
   public void changeSeedScore(int score,Node V) {

   }
}
```

2.5 Right_handside.java code:

```
package Business_Objects; public class
Right_HandSide {

   public void rightHandSide(int i) {

   }
   public void createActive(int score) {

   }
   public void createEdges(int score,Node E,Node U) {

   }

   public void checkPaths(int score,Node V) {

   }

   public void updatePaths(int score,Node v) {

   }
   public void updateSeedScores(int score,Node U) {

   }
   public void deleteActivePoint() {

   }
   public void mergeEdges(int score,Node A,Node U) {

   }
   public void transferEdges(int score,Node U,Node E) {

   }
   public void copySeedRHS(Node V,Node U) {

   }
   public void updateNodes(int score,Node U,Node E) {

   }
   public void updateEndArray(Node E) {

   }
}
```