Finding Surprisingly Frequent Patterns of Variable Lengths in Sequence Data

by

Reza Sadoddin

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science
University of Alberta

Abstract

We address the problem of finding 'surprising' patterns of variable length in sequence data, where a surprising pattern is defined as a subsequence of a longer sequence, whose observed frequency is statistically significant with respect to a given distribution. Finding statistically significant patterns in sequence data is the core task in some interesting applications such as Biological motif discovery and anomaly detection. We investigate the problem of 'redundant patterns', where the presence of few 'true' anomalous patterns in the data could cause a large number of highly-correlated patterns to stand statistically significant just because of those few anomalous patterns. Identifying 'true' anomalies in a set with many 'redundant patterns' can be challenging. Our approach to solving this problem is based on capturing the dependencies between patterns through an 'explain' relationship where a set of patterns can explain the statistical significance of another pattern. The 'explain' relationship allows us to address the problem of redundancy by choosing a few 'core' patterns which explain the significance of all other significant patterns. We propose a greedy algorithm for efficiently finding an approximate *core* pattern set of minimum size. To extend the utility of our method to a broader class of applications, the proposed framework is generalized by allowing the 'surprising patterns' to represent a class of subsequences with a certain amount of variation w.r.t a *core pattern*. Using both synthetic and real-world sequential data, chosen from different domains including Medicine, Computer Security, and Bioinformatics, we show that the proposed notion of *core* patterns very closely matches the notion of 'true' surprising patterns in data. We also compare our method with five other well-known anomaly detectione techniques. The results show a better matching of our predictions with the ground truth compared to those of our comparison partners. When compared with 14 well-known methods on the interesting application of the Biological motif discovery, and on a widely-used benchmark, our proposed method achieves better or comparable results in finding *motifs*, a special case of our surprising patterns.

Acknowledgements

It is my great pleasure to thank and appreciate the awesome people who have given me help, support, encouragement, and guidance during my journey as a PhD student.

I was very very lucky to be supervised by two of the greatest people and professors in the Department of Computing Science, professor Joerg Sander and Davood Rafiei. Joerg Sander and Davood Rafiei were incredibly patient and supportive during all this time, not only in my research activities when I was encountering obstacles which resulted in slow progress in some periods of my research, but also in my life and career challenges. They were very understanding and helpful during the time I had to work remotely because of my personal life. Joerg Sander has contributed substantially to this thesis by bringing new ideas and directions, constructive criticism, getting involved in details when necessary, and careful reviews of the manuscripts and the thesis generated from this research study. Davood Rafiei has contributed a lot during research work by asking thoughtful questions, encouraging to get deeper in problems, bringing new ideas for application of our research work, and being very responsive in reviewing the manuscripts. I was supported jointly by Joerg and Davood during most of my PhD study.

I would like to thank my examiners, professor Lukasz Kurgan and professor Guohui Lin for giving me useful feedback during my candidacy examination. Dr. Lukasz Kurgan encouraged me to extend the proposed methodology to be applicable on the Biological motif discovery, and Dr. Guohui Lin helped me to investigate some theoretical aspect of the algorithms, which resulted in the NP-Complete proof of the main problem in this thesis. Also, I would like to thank Denilson Barbosa for reading my candidacy proposal and providing me with valuable suggestions. Thanks to Proferssor Jian Pei for giving me the pleasure to be an external member of my committee. I am also thankful to Ian Parsons from Syncrude for attending our regular biweekly research meetings, and giving valuable insights and suggestions in my research work.

During my PhD study in Edmonton, I had the chance to meet some of my best friends in life. I am always proud of my lab-mates in the Database lab, Pirooz Chubak, Vahid Jazayeri, Mojdeh Jalali, and Davoud Moulavi, with whom I have a lot of good memories. Also, I would like to thank my wonderful friends Yavar Naddaf and Fariborz Kiasi who were great companions during my PhD study in Edmonton.

My parents, and my two sisters are my greatest treasure in the life, from whom I have always received tremendous support and unconditional love. They have been encouraging and inspiring me from the first day I entered the primary school and all the time during my long journey in Academia. My deepest thank and warmest

appreciation goes to my family. Also, it is my great pleasure to thank my girlfriend, Elaheh Ehsani, for helping me wrapping up my PhD thesis, especially in the last year of my PhD study when I was working as a full-time employee.

This research was enabled in part by support from Westgrid (www.westgrid.ca) and Compute Canada Calcul Canada (www.computecanada.ca).

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sequence data are found in a wide variety of application domains such as Computer networks, Bioinformatics, Web applications, Financial transactions, Weather measures, Medicine, *etc.* The volume and diversity of sequence data has been increased as new applications have been emerged (e.g. high-throughput DNA sequencing, Web clickstream analysis) or traditional applications have been made available online (e.g. online shopping stores). The sequence data has been the target for a diverse set of data mining tasks, where the main goal is to analyze the data from different perspectives, discover *interesting patterns* in the data, and summarize it into useful information. These tasks encompass a wide range of spectrum, from more traditional *association rule mining* and *frequent pattern mining* tasks to newer ones, such as intrusion detection, customer behaviour analysis in online stores, and Biological motif discovery.

Core to a data mining task on the sequence data is the type of patterns that can be discovered, and the characteristics of an *interesting pattern*. While in the traditional data mining problems the patterns were characterized based on user-defined and domain-dependent parameters (*e.g. support* in frequent pattern mining), the newer problems define *interesting patterns* based on the data characteristics itself (*e.g. Chi-square* statistics [100]). In general, it will be appealing to develop methods which require less input parameters or behave more robust with respect to input parameters.

Our research concerns with finding *unexpectedly frequent patterns* of variable lengths in a long sequence data. We call these patterns *surprising* because of the fact that they are observed more frequently than one would *expect* under normal conditions. Finding *surprising* patterns in sequence data is a key problem in many applications domains as diverse as Bioinformatics, Computer security and medicine. In Bioinformatics, the surprising patterns, often referred to as "motifs", are believed to have some important biological significance and regulate gene expressions [27]. *Motif discovery* in this domain is the problem of finding subsequences in a DNA sequence that are *overrepresented* relative to a background distribution. In Computer security, surprising patterns may correspond to a sequence of commands or system calls executed by an attacker or a malicious program [37]. In time series data, a surprising pattern might correspond to a "discord", which has been defined as a subsequence in a longer time series that is of maximal distance to its nearest neighbour(s) and is shown to capture anomalies in ECG data (*e.g.* heart arrhythmia)

and Space telemetry (e.g. failures in spacecrafts) [23, 58].

1.1 Motivating Applications

In the following, we review two important applications of finding *surprising* patterns in sequence data, including the Biological motif discovery and anomaly detection in sequence data.

1.1.1 Biological Motif Discovery

A *motif* is a nucleotide or amino-acid sequence pattern that has some biological significance, such as being DNA (deoxyribonucleic acid) *binding sites* for a regulatory protein, *i.e.*, a *transcription factor*. Transcription factors are proteins that bind to specific DNA sequences and regulate the *gene expression* by activating or inhibiting the transcription machinery. *Gene expression* is the process by which information from a gene is used in the synthesis of a functional gene product, such as proteins.

Understanding the procedures that regulate *gene expression* is a major challenge in biology. A key part of this task is to find the regulatory elements, specially the binding sites for *transcription factors*, or *motifs*. This is a necessary first step in determining which factors regulate the *gene* and how. Also, finding the same motifs in multiple genes' regulatory regions suggests a regulatory relationship amongst those genes. Due to biological significance and functions assigned to these regulatory elements, they occur multiple times in the same genome, and they are very likely to be conserved during the evolution. There is an important hypothesis that the transcription factor binding sites are "over-represented" in the regulatory regions of DNA sequences [26]. A statistically *overrepresented motif* means a motif that occurs more often than one would expect by chance.

A practical consequence of this hypothesis is that plenty of computational motif discovery tools have been developed to find these regulatory elements. These algorithms search for overrepresented motifs in this collection of regulatory sequences. Efficient computational tools can potentially provide high-quality prediction of binding sites and reduce the time needed for experimental verification. The assumption behind the motifs is very similar to that of the 'surprising patterns' in our research study. As discussed already, and will be defined more formally in Section 3.5, the *surprising patterns* are subsequences which are statistically significant with respect to a background distribution. The proposed methods in our research study provide a means for finding motifs in Biological sequences, and particularly address the issue of the motif lengths.

However, this apparently simple approach for finding motifs based on the over-representation assumption is complicated by several factors. First, most of binding site motifs are short patterns (5 to 20 base pair long) in the presence of a great amount of statistical noise (a typical input might be a regulatory region of length 1000 of each gene). More importantly, there are usually some variations between the binding sites of a transcription factor, in the form of insertions, deletions, and mutations. Also, the nature of these variations is not unknown. Example of binding sites for the hypothetical protein *BUH* and human genome are shown in tables 1.1 and 1.2. As it can be observed, the amount of variations in some cases can be in

the level of just mutations (*e.g.* Table 1.1), and in other cases, the binding sites can be of different lengths with lots of variations (*e.g.* Table 1.2), making it harder for computational tools or even domain experts to identify them.

Table 1.1: List of binding sites for the hypothetical protein *BUH* (adapted from [29])

ATGACATCAT
ATGCTGCCAA
ATGCGATAGG
ATCGACGTAC
ATTCGCTAAT
GCTAGCTCAC
TAGCTAGCAT
ATCGCGCCAT
ATCGCTACAT
ATTGCGAGAT
ATGCTGATAT
TTGTGATGAT
TTAGCATGCC
ATTGCATCAG
GTGTGATCAT
GTACTGACAT
ATGCATTCAG
ATGAATACTT
ATGGACCCCT
ATGGACCCCT
ATGTTGCCAG
ATGAGATTAT
ATCAGACCAT
ATGACAGCAT
ATGATGACTT

1.1.2 Anomaly Detection in Sequences

Anomaly detection is "the identification of items, events or observations which do not conform to an expected pattern or other items in a dataset" [18]. Anomaly detection has applications in a wide variety of applications such as fraud detection for credit cards, insurance or health care, intrusion detection for cyber-security, fault detection in safety critical systems, and military surveillance for enemy activities. Anomalies are also referred to as *outliers*, *novelties*, *noise*, *deviations* and *exceptions*.

Due to wide range of applications of anomaly detections methods, detection methods could be different based on the different aspects of the problem. This includes the nature of the input data (*e.g.* categorical, numerical), type of the anomaly (*e.g.* point-based anomalies, collective anomalies), operation mode of the detection method (*e.g.* supervised, unsupervised), and the output of an anomaly detection method (*e.g.* score, label).

When dealing with anomalies in sequence data, the following scenarios are possible in practice:

Table 1.2: List of binding sites for genome belonging to Human [109].

ATGACATCAT
TGACGTCA
TGCGTCA
GTGACATCAT
ATGAGTCAGA
GCCTGCGATGATTTATACTCACAGGA
CTAAGGGGTCA
GTGACTAA
TGAATGACTCACCTTGGCACAGACACAATGTTCGGGGTGGGCACAGTGCCTGCT
GGCATAATGGGTCTGTCTCATCGTC
CCAAGCGTGACTGC
TGAATCA
TGCGTCA
ATGAATCATC
GCAAGGATGAGTCAAGCTGCGGG
TGAGTAA
ATGAATCAT

- *Detecting anomalous subsequences in a long sequence*: In this scenario, a long data sequence is given which represents the normal behaviour of a process, except possibly at some locations, where the normal process is intervened by anomalies. The goal is to detect subsequences which are different from the rest of the sequence. An example of this scenario is to detect anomalous intervals in an ECG recording of a patient which might correspond to *heart arrhythmia*.

- *Detecting anomalous subsequences in a set of sequences*: In this scenario, a database of sequences are given and the goal is to find the sequences which *deviate* from the rest. Although the sequences are not generally of the same length, but the assumption is that they capture the same process. A representative example is the motor power recordings of an industrial device which is doing a repetitive task, in which the recording of the power in each period results in a time series sequence.

As it was discussed, the anomaly detection in sequence data has many variations, with wide range of applications in different domains. Our research study concerns with the first scenario discussed above, where the goal is to find anomalous subsequences in a long sequence data. In terms of the nature of input data, our main focus is on symbolic data. Even though the proposed method in our research work can be applied on time series data after a discretization, we have not evaluated our proposed method on such input. We assume the presence of a *reference* data, which represents the normal behaviour of the process, and can be used for learning the parameters of the model. Also, our proposed method assigns a final label to each position of the sequence in the format of *anomalous* or *normal* without requiring any additional parameter to make predictions based on the anomaly scores.

1.2 Challenges

Most of the previous works for finding surprising patterns (*e.g.* transcription factor binding sites in DNA sequences, anomalies in long sequence data) expect the user to provide the length of the anomalous patterns. Finding *surprising* patterns without knowing their lengths poses several challenges. The length parameter is not intuitive and difficult to set, in many applications. For instance, motifs in DNA sequences typically vary between 6 to 20 bp in length, but they could also be 100-200 bp in length. Without knowing the true length(s), one should run the particular algorithm exhaustively, each time for a specific length. Our study shows that running a length-dependent method with different lengths of patterns will result in a large number of "highly-correlated" patterns. The *correlation* is due to the fact that the *overlap* between patterns results in similar outcomes (e.g. anomaly score) for partially similar patterns. The set of patterns generated using theses methods is typically so large that the true anomalous patterns in the data cannot be revealed easily. We refer to such presence of highly correlated patterns as the "redundant patterns" problem. The problem of redundancy becomes more crucial when there are more statistically significant patterns in a sequence or there is a degree of variation in different instances of *surprising patterns* . This results in a significantly larger number of correlated patterns with *true* surprising patterns standing out as *significant*.

The second drawback of previous methods is the robustness of their results. The definition of *surprising* patterns and the required parameters have a large impact on the convenience of use and the robustness. For instance, a true surprising pattern might be observed a few times in a sequence. A definition that compares the observed frequency of a pattern (*i.e.* the number of times a pattern occurs in a sequence) with a threshold might or might not capture the pattern based on a selected threshold value. The proposed methods for finding surprising patterns in the literature depend on different input parameters. An example is the neighbourhood parameter K for KNN-based anomaly detection methods [20]. Some of the methods for approximate frequent patterns depend on the support (frequency) threshold to consider a subsequence as a candidate surprising pattern [35, 126]. These parameters are in addition to the parameter "length" of the patterns, which largely impacts the effectiveness of these methods. Coming up with suitable parameter settings becomes more challenging when multiple parameters are involved. Our experiments show that different parameter settings produce largely deviating results, making these methods less reliable in a real setting where the best parameter settings are unknown.

Our work addresses the aforementioned problems by eliminating the need for non-intuitive input parameters (such as length, frequency threshold, *etc.*) and producing robust results represented by a concise, non-redundant set of relevant patterns. Another important factor in the utility of a *surprising* pattern finding technique is the variations allowed in the signature of a *surprising* pattern. In many applications, such as motif discovery, a surprising pattern (*e.g.* motif) is characterized by a group of subsequences which look 'similar' to each other noticeably, with a degree of 'variation'. Most traditional anomaly detection methods [20] (*e.g.* t-STIDE, HMM, KNN) which are based on a simple string representation of pat-

terns allow no variations among pattern occurrences, hence they fail to detect these *generalized patterns*.

1.3 Contributions of the Thesis

The contributions of this thesis can be summarized as follows:

1. We provide a domain-independent formulation of the problem of finding surprising patterns in sequence data based on statistical hypothesis testing, which does not require the *length* of the patterns as an input parameter.

2. We investigate the problem that a few embedded anomalous patterns can lead to a large number of "redundant" patterns, from which the 'true' anomalous patterns cannot be revealed easily.

3. We propose a statistical method that captures an "explain" relationship where a set of patterns can explain the statistical significance of another pattern.

4. Using this "explain" relationship, we address the problem of redundancy by choosing a few 'core' patterns which explain the significance of all other significant patterns. The "explain relation" is an important statistical tool which allows to us to characterize the solution to finding 'true' surprising patterns as a minimum set of patterns which can explain the frequency of all the other (redundant) patterns in the data

5. We propose a greedy algorithm for efficiently finding a minimal *core pattern set* as an approximate solution.

6. We extend the basic model (*i.e.* the model based on a string representation of patterns) in order to capture patterns with a degree of variation (*i.e. approximate patterns*). The proposed techniques in previous sections are also extended to find these approximate surprising patterns and to cope with the model complexity.

7. We analyze the time complexity of our proposed method based on the basic model and the extended approximate matching model.

8. We investigate strategies to improve the running time of our proposed method. We develop a new strategy for speeding-up the exact p-value computation, and using approximate p-values as a filtering step to avoid costly exact p-value computations. We investigate the properties which characterise the correlation between significant patterns. This allows us to speed-up the construction of a *core pattern set* by limiting the score of 'correlation' tests for a pattern. Using these properties, the computationally expensive correlation tests are performed just for patterns which are likely to be correlated with respect to a given pattern. In addition, a *Prefix tree* data structure is used to speed-up heavy probability computations for patterns based on the extended approximate model. We develop a parallel architecture of our framework in the Westgrid cluster, which results in a significant speed-up in running time of our proposed method by leveraging the parallelism capabilities of computationally expensive components of our framework.

9. Finally, we evaluate the performance of our methodology on both synthetic and real-world data. The synthetic data is generated based on a Random walk model which is used as a modelling some real-world processes such as time series of the financial markets and dynamics of the Blogosphere [95]. The settings allow evaluating important aspects of a *surprising* pattern finding algorithm, such as the variations in the length of the patterns, and the deviation of the patterns from the background distribution, by changing the parameters of the model. The real-world datasets are chosen from three different domains including the Medicine, Computer Security, and Bioinformatics, which represent three interesting applications of our method. Compared to the general anomaly detection techniques, our proposed method based on the basic model (*i.e.* exact pattern matching) achieves a higher matching with the 'true' anomalous patterns on the synthetic, ECG and Masquerading users datasets. Our extended method for fining approximate significant patterns is evaluated on a well-known motif discovery benchmark, and is compared with 14 well-known motif finding algorithms from the domain of Bioinformatics. In 2 out of the 3 experimental studies, our proposed method performs better than or as good as the Weeder motif finding algorithm (the best performing method on the this benchmark).

1.4 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we review the most relevant research works to our work, including the works on anomaly detection in sequence data, finding Biological motifs, subspace clustering, motif discovery in sequence data, and frequent pattern mining in sequence data. We present a problem formulation and demonstrate the challenges of the problem using a synthetic dataset in Chapter 3, after giving a background on basic concepts used in our methodology, including the statistical significance testing, Markov chain model, and smoothing techniques for Makov models. In Chapter 4, we provide a more formal definition of our problem through a *core pattern set* which characterises the 'true' significant patterns in the data, and present the definitions which characterise the correlation between statistically significant patterns. In Section 4.1, we present an exact formula to compute the correlation between significant patterns by computing the p-values of patterns in a long sequence data in the presence of some constraints. In Section 4.2, we propose an algorithm for constructing an approximate *core pattern set* as the solution to our problem. A more generalized definition of *surprising patterns* is given in Chapter 5 and new algorithms are derived for computing the p-value of new motif models and capturing the correlation between the extended patterns. We analyze the time complexity of our method in Chapter 6. In Chapter 7, we evaluate the performance of our proposed method on synthetic and real-world datasets selected from the domain of Medicine, Computer Security and Bioinformatics. Finally, we discuss the strengths and weaknesses of our work and propose some ideas for further research in Chapter 8.

Chapter 2

Related Work

2.1 Anomaly Detection in Sequence Data

Anomaly detection is a broad research topic with applications in different domains and the related work in the literature differ from each other based on the type of the defined anomalies, nature of the input data, and the proposed techniques. The proposed method in our research work can be used for detecting a class of anomalies in the sequence data, in which the anomalous subsequences are observed more than their *expected* frequencies.

Four main aspects of the anomaly detection problem can be summarized as follows [18]:

1. **Nature of Input Data**: The input data can be of type *categorical* or *numerical*. In the case of categorical, values or observations can be sorted into groups or categories [28]. The eye color, gender and the swimming level are examples of categorical data. In numerical data, the values or observations can be measured, and can be placed in ascending or descending order [28]. Height, time, and temperature are examples of the numerical data. The numerical data can be further divided into the *discrete* and *continuous* categories. A set of data is said to be *discrete* if the values belonging to it are distinct and separate (*e.g.* the number of people in a room), whereas in *continuous* data the observations may take on any value within a (finite or infinite) interval (*e.g.* the time required to run a mile) [28]. In addition to the type, each value or observation might represent a *single* attribute or multiple attributes, which introduces the *univariate* or *multivariate* sequences, respectively. The nature of input data affects the applicability of anomaly detection techniques. For example, for statistical techniques different statistical models have to be used for *numerical* and *categorical* data (although discretization is an alternative to convert numerical data to categorical data, this does not change the different nature of applicable methods. Moreover, there is no guarantee that the interesting features in the original numerical data format will be preserved after transformation). Similarly, for nearest neighbour-based techniques, the nature of attributes would determine the distance measure to be used. The common distance measure used for numerical data is *Euclidean* distance, whereas categorical attributes require overlapping-based metrics such as *hamming* distance.

2. **Type of Anomaly**: Anomalies can be divided into two main categories based on their deviation from the normal behaviour: 1) *Point-based* anomalies are those anomalies in which data instance can be considered as anomalous with respect to the rest of data. This is the simplest type of anomaly and has been the focus of majority of previous research on anomaly detection. An example of this type of anomaly is a credit card transaction for which the amount spent is very high compared to the normal range of expenditure for the card holder. 2) *Collective* anomalies are types of anomalies in which a collection of related instances is anomalous with respect to the entire dataset. An example of a collective anomaly is a sequence of *port scans* from a remote machine against a network, followed by *ssh*, *buffer-overflow*, and *ftp*, which could be indicative of a web-based attack. In this case, the observed data points individually are not typically indicative of anomalous behaviour.

3. **Data Labels**: Based on the availability of the labels for *normal* or *anomalous* behaviour, anomaly detection techniques can be grouped under one of the two schemes:

   (a) *Supervised anomaly detection*: The main assumption in this mode is the availability of a training data set, which has labeled instances for normal and anomaly classes. Predicitve modeling techniques, mainly based on classification, can be used to train models from the provided labeled data and classifying unseen instances.

   (b) *Unsupervised anomaly detection*: Techniques that operate in unsupervised mode do not require training data, and thus are most widely applicable. However, the implicit assumption behind the techniques in this category is that the data is dominated by normal instances and anomalies are just exceptions in the test data. If this assumption is not true, these techniques will result in high false alarm rate.

4. **Output of Anomaly Detection**: An anomaly detection technique can produce results in the form of *Score* or *Label*. *Scoring* techniques assign an anomaly score to each instance in the test data depending on the degree to which that instance is considered an anomaly. Thus the output of such techniques is a list of instances sorted based on their *anomaly scores*. A user-dependent threshold is usually required in this case to either consider top K anomalous instances or select the anomalies based on a cut-off threshold. *Labeling* techniques, on the other hand, produce a deterministic outcome with a label (e.g. *anomaly*, *normal*) based on an internal mechanism for classification.

The anomaly detection techniques can be divided into five main groups [19], including the *Window-based* techniques, *Markovian* techniques, *Hidden Markov Model-based* techniques, *Similarity-based* techniques, and *time series-based* anomaly detection techniques.

2.1.1 Window-based Techniques

Window-based techniques are generally based on extracting windows of a fixed length using a sliding window technique from a test sequence, and assigning an

anomaly score to each window. This approach can be used to identify *anomalous* susbequences in a long sequence data, or finding entire test sequences which are *anomalous* by aggregating the *anomaly scores* of its windows. These techniques are particularly useful when the cause of anomaly can be localized to one or more shorter substring within the actual sequence [37].

In a windows-based technique, a training data and a length k of the window is given as input. The general scheme for a window-based technique is as follows: in the training phase, k-length sliding windows are extracted from all training sequences and the frequency of occurrence of each unique window in the training data set is maintained (as a normal *profile*). In the test step, sliding windows of length k are extracted from the test sequence. An anomaly score can be assigned to each window by comparing the *observed* frequency of the sequence enclosed by the window to its *expected* frequency, which is derived from the training data. More formally, for a pattern A which has been observed t times in the test sequence S_{test}, and e times in the training sequence S_{train}, the *anomaly score* can be derived using the following:

$$\text{Anomaly Score} = \frac{t \times |S_{train}|}{e \times |S_{test}|} \tag{2.1}$$

where $|S|$ represents the length of a sequence S. This gives us a means to locate *anomalous* subsequences in a long sequence of data using an anomaly threshold λ. Alternatively, we can aggregate (e.g. compute the *average*) the anomaly scores of windows along the sequence and compute the anomaly score of the entire sequence.

The basic window-based technique has been originally used for operating system call intrusion detection in a system called *t-STIDE* (Threshold-based Sequence Time Delay Embedding) [52, 112]. Different variants of the t-STIDE method have been proposed in literature [15, 43, 44], differing from the basic method in how an *anomaly score* is calculated for a window, and which aggregation functions are used to compute the anomaly score over the entire sequence from window's scores.

Analysis and Comparison to Our Research: The Window-based techniques are simple to implement, and can be used to identify the anomalies which are local to specific regions in a sequence. However, storing the profile of normal subsequences requires large memory. Also, the windows-based techniques are highly reliant on the size of the window that is chosen. If the size of the window is too small compare to true anomalies in the sequence, all the selected windows have a high probability of occurrence in the training data, and therefore a large number of anomalies might be missed by a technique in this category. On the other hand, if the size of the window is too large, every subsequence in the test sequence might be detected as an anomaly, and will result in a highly-redundant solution. The techniques in this category are similar to our proposed method for finding surprisingly frequent patterns in the sense that we also use sliding windows to extract patterns and use an anomaly notion that is based on frequency of the patterns. However, we address the issue of choosing the right size of the window by finding the *core* patterns in the data. The issue of the redundancy will be addressed automatically as a result of selecting the right length.

2.1.2 Markovian Techniques

The *Markovian* techniques approximate the 'true' distribution of the data using a *Markov Chain* model, in which the probability of occurring a symbol in a sequence depends on the last m symbols in the sequence, where m is the order of the *Markov chain* model. More formally, the probability of observing a symbol a_i, given the sequence observed so far can be computed as follows:

$$P(a_i|a_1a_2...a_{i-1}) = P(a_i|a_{i-m}a_{i-m+1}...a_{i-1}) \tag{2.2}$$

In the training phase, the parameters of the Markov chain model of order m are derived by counting the number of occurrences (*e.g.* frequencies) of all the subsequences upto length m. The conditional probabilities of a symbol a_i after a subsequence of length at most m can be derived using the *Markov* chain rule. The simplest form of computing the conditional probability of a symbol a_i followed by a subsequence $a_{i-k}a_{i-k+1}a_{i-k+2}...a_{i-1}$ is as follows:

$$P(a_i|a_{i-k}a_{i-k+1}a_{i-k+2}...a_{i-1}) = \frac{f(a_{i-k}a_{i-k+1}a_{i-k+2}...a_{i-1}a_i)}{f(a_{i-k}a_{i-k+1}a_{i-k+2}...a_{i-1})} \tag{2.3}$$

where $f(W)$ is the frequency of the string W in the training sequence S_{train}.

Different variants of this basic Markovian model have been proposed in the literature, based on the way the conditional probabilities are computed in the interest of space required for storing the frequencies, and also the methods used for addressing the sparsity of the data.

For an alphabet size of $|\sum|$, and Markov chain model of order m, the total number of frequencies required by the basic technique will be $(|\sum|-1)|\sum|^m$. This could lead to huge amount of space (exponential in the order m of the model), and some research works have addressed this issue. One such a model is proposed by Michael *et al.* [77]. The proposed model, called the *Extended Finite State Automata* (*EFSA*), by Michael *et al.* is like a traditional *Finite State Automata* (*FSA*), but with frequencies associated with the nodes and the number of transitions from one node to another. Each node of a EFSA corresponds to a substring of length k, and there is a transition from node A to another node B if the suffix of length $k-1$ of node A is equal to the prefix of length $k-1$ of node B. Only those nodes and transitions that are observed in the training sequences are stored in the EFSA. Thus, the size of EFSA is typically smaller than the total possible size of $(|\sum|-1)|\sum|^m$. Marceau *et al.* propose a similar method which utilizes a *suffix tree* for storing the substrings and their frequencies, and a FSA for state transition of the model [75].

Another challenge in building *Markovian* models is the *data sparsity*. The estimation of conditional probabilities based on the formula 2.3 is reliable if sufficient amount of training data is available to provide a realistic estimation of frequencies of substrings in practice.

Example 1 *Assume that the substring "aabbb" occurs once across all training sequences and is followed by symbol b in that single occurrence. The Formula 2.3 will assign a conditional probability of* 1 *to* $P(b|aabbb)$*. However, this conditional probability might not be reliable considering the fact that the occurrences of other symbols after the subsequence "aabbb" might be possible in practice, but don't happen to be observed in the training data. This poor estimation might give undesirable results.*

One solution to addressing this problem is to allow symbols to be conditioned on a variable length of history subsequences. For the example 1, the probability $P(b|aabbb)$ can be substituted with $P(b|aabb)$ if the probability associated with history subsequence "abbb" is more reliable based on a given pruning criterion (*e.g.*, frequency of the subsequence"abbb" is greater than a certain threshold). The *Probabilisitic Suffix Trees* (*PSTs*) [93] and *Interpolated Markov Models* (*IMM*) [47] have been utilized to efficiently compute the variable length conditional probabilities of a symbol.

Another solution to the sparsity problem is to estimate the conditional probability of a symbol a based on the previous m symbols which are not necessarily contiguous or immediately preceding to a. In other words, the symbols are conditioned on a sparse history. Referring to the example 1, if the sequence "aabbb" occurs rarely in the training data, the conditional probability $P(b|aabbb)$ can be replaced with $P(b|aXbXb)$ where X can be replaced with any symbol of the alphabet. Eskin *et al.* propose the *Sparse Markov Transducers* (*SMT*s) model based on the idea of sparse history and extending the *probabilistic suffix trees* (*PST*s) to allow wild cards in the history subsequences [30].

Lee *et al.* utilize the *RIPPER* rule learner [24] to address the issue of data sparsity [65]. The main idea behind their method is to model the task of probability estimation of a symbol a_i based on a history subsequence of length k with a classification technique in which every symbol in the history subsequence is treated as an input feature and the symbol a_i is treated as the target label. The *RIPPER* is used to learn rules from this categorical data set to predict the $(k+1)$th symbol given the first k symbols. In the test phase, the first rule that matches the history subsequence $a_{i-k}a_{i-k+1}...a_{i-1}$ is chosen. If the target of the selected rule is a_i, then the probability $P(a_i|a_{i-k}a_{i-k+1}...a_{i-1})$ is set to 1. Otherwise, the inverse of the confidence associated with the selected rule is assigned to the conditional probability.

Analysis and Comparison to Our Research: The *Markovian* techniques take advantage of probabilistic models to build a normal profile from the training data, which can result in more efficient memory consumption using data structures such as probabilistic suffix trees. The techniques in this category use a similar underlying model (Markov model-based) to capture the temporal nature of sequences and computing the probabilities of occurrence of sequences under the model. However, the *Markovian* techniques also rely on the window length that is chosen for extracting the subsequences and estimating their likelihood. Therefore, the techniques in this category suffer from the same problems of the Window-based techniques in missing partial anomalies or producing large number of redundant anomalies based on the length of the chosen window. The techniques employed by the *Probabilisitic Suffix Trees* (*PSTs*) [93] and *Interpolated Markov Models* (*IMM*) [47] to use a variable length of the history in estimating the probability of occurrence of a symbol is more helpful in addressing the issue of data sparsity rather than finding the true length of the anomalies. Chandola *et al.* [18] compare a fixed *Markovian* technique with a variable and a *sparse* Markovian technique on data from different application domains and show that the fixed Markovian technique (using EFSA) outperforms the variable (using *PST*) and the sparse (using *RIPPER*) techniques on many data sets. The same comparative study also suggests scenarios in which the *variable* and *sparse* Markovian techniques can perform better than the *fixed Markovian* tech-

niques. In our proposed method, we find the length of 'true' anomalies by selecting a *core* pattern set which can explain the significance of other overlapping patterns, and avoid the problem of redundancy.

2.1.3 Hidden Markov Model-based Techniques

A *Hidden Markov Model* (HMM) is a statistical *Markov* model in which the system being modeled is assumed to be a *Markov* process with unobserved (hidden) states. In simpler *Markov* models (like a *Markov chain*), the state is directly visible to the observer, and therefore the *state transition* probabilities are the only parameters. An HMM is parameterized by a hidden state transition matrix and an observation (*emission*) matrix.

The simplest way to use an HMM for anomaly detection is use the training data to learn an HMM that best describes the normal process that generates the data. The learning step can be done using the *Baum-Welch* algorithm [10], in which the transition and emission probabilities can be derived using the training data. In the test phase, and for each given test subequence, the probability of occurrence of it can be estimated using the *Forward* algorithm. Lane has utilized this approach to identify anomalous computer usage in system call data corresponding to user behaviour [62]. Yamanishi *et al.* use a similar approach for identifying anomalies in system log data [119], in which a mixture of HMMs are used to model the normal sequences.

Other variations of the HMMs have been applied for anomaly detection by analyzing the most likely or optimal hidden state transitions from the training data, which can be derived using the *Viterbi* algorithm [36]. Once the most likely state transitions are derived for a test datasets , any sequence anomaly detection technique (that has been discussed in previous sections) can be applied to find anomalies in the test sequence. As an example work, Qiao *et al.* apply a window-based technique on the transformed hidden state sequence [118].

Analysis and Comparison to Our Research: The HMM-based techniques are similar to our research work in that they can be used for finding anomalouse subsequences in a long sequence data. The HMM-based techniques are able to model complex systems, even in the case that the normal observation sequences are significantly different from each other. A comparison performed by Forrest *et al.* shows that the HMM-based methods perform comparable to or better than the Windows-based and Markovian techniques for detecting anomalies in system call sequences [112]. However, deriving the parameters of an HMM is computationally expensive. This can be a real obstacle in the utility of these techniques in practice. In our experiments, we had to abandon the model training in some cases when the training was not terminated after a day. Moreover, the HMM-based techniques suffer from the same problem of relying on the length parameter which were attributed to the previous categories of techniques.

2.1.4 Similarity-based Techniques

The main idea behind *similarity-based* techniques is to use a distance measure to compute the similarities of test sequences, or their subsequences, to the sequence(s) in the training data. An *anomaly score* can be derived from the inverse of similar-

ities. The general scheme for a similarity-based technique is as follows: In a test phase, the similarity of a test sequence, or its subsequences, to each sequence or subsequnce in the training set is computed. The similarities are then *aggregated* to derive a similarity measure between the test sequence and the training data. The *anomaly score* is the inverse of the similarity.

The K-th Nearest Neighbour (KNN) method is the most common aggregation function which combines the similarities by selecting the K-th most similar sequence in the test data and its inverted distance as the aggregated similarity [20]. Clustering can be used to avoid point-wise distance measure computations and improve the running time during *test* phase. Budalakoti *et al.* [13, 14] utilize a clustering-based technique in which the training sequences are first clustered into a fixed number of clusters using the *K-medoid* algorithm [56]. In the test phase, the anomaly score of a test sequence is computed based on the distance to its closest *mediod*.

Distance computation is a core part in every similarity-based method. The most common similarity measure of two discrete subsequence is the *Simple Matching Coefficient* (SMC) [99], which computes the similarity of two subsequences of equal length as the number of positions in which the two subsequences match. Other similairty measures have been proposed which do not require the subsequences to be of equal length. The *nLCS* defines the length of the longest common subsequence within two sequence s_i and s_j as their similarity and has been used in several anomaly detection techniques [13, 14, 20]. Liu *et al.* use a variation of the *edit distance* to find anomalies from a database of protein sequences corresponding to different organisms [39].

Analysis and Comparison to Our Research: The techniques in this category are highly reliant on the parameter K, in addition to the parameter length, in detecting the anomalies, and this is not an intuitive parameter to be provided by a user. Our experimental study show that different values of K usually result in highly deviating results. This input parameter is in addition to the parameter length, which has to be provided by the user, and results in a highly redundant set of anomalous patterns without addressing the dependencies between the overlapping patterns.

2.1.5 Anomaly Detection in Time Series Data

Dealing with time-series data is different from the discrete data in many aspects mainly due to the fact that continuous values in a time series sequence makes it challenging to make direct correspondence between "states" and a limited number of symbols. Moreover, different distance functions are required to capture the similarity between time series sequences.

Keogh *et al.* introduce the problem of finding 'discords' in a long time series sequence [58]. Discords are defined to be "subsequencs of a longer time series that are maximally different from the rest of the subsequences" [58]. In this work, first all k-length windows are extracted from the given sequence S and stored as a database of fixed-length windows. In the second step, each window is assigned an *anomaly score*, which is the distance to its K^{th} nearest neighbour in the database. The subsequences with highest anomaly scores can be returned as *top discords*.

Variations of the basic discord discovery algorithm have been investigated in the literature by using different measures for assigning anomaly scores to subsequences,

and making efforts to improve the running time of the basic algorithm. The *Window Comparison Anomaly Detection* assigns an anomaly score to each subsequence using an information theoretic measure called *Compression-Based Dissimilarity* (*CDM*) [60]. The main idea behind CDM is that a subsequence is assigned a higher anomaly score if excluding it will lead to a significant compression in the remaining of the time series sequence. Wei *et al.* compare the bitmap of the current window with the bitmap of the previously adjacent window to derive an anomaly score for the susequence in the current window [115].

Naive computation of the basic discord discovery is quadratic in the length of the time series sequence. One general scheme for reducing the time complexity is based on computing the *anomaly score* of subsequences which are more likely to be discord. A simple way of implementing this idea is to abandon the distance computations for a subsequence if at any point of calculation its distance to its current K^{th} nearest neighbour becomes lower than the anomaly score of current m^{th} discord (assuming that we are interested in top m discords). This technique and its variations have been used to the problem of discord discovery in several research works [70, 114]. Since a vast majority of windows tend to be normal, the proposed technique based on early abandonment has the potential for substantial pruning, especially if the anomalous windows are discovered early.

Analysis and Comparison to Our Research:

The techniques in this category are similar to our work in that they can be used for detecting anomalous subsequences in a long sequence. However, most of the anomaly detection techniques in time series data focus on finding anomalies based on their deviation from their nearest neighbours, rather than the deviation from the *expected* frequencies. This makes the techniques in this category different from our research work. Moreover, the techniques in this category target the continuous time series data, and are different from our research which focuses on the symbolic data. Even though the models and the techniques proposed in our research can be potentially used for finding anomalies in the time series data after transforming it into a symbolic sequence, the original features in the time series data might not be maintained after transformation. Therefore, finding anomalies in the time series data and the symbolic data require different treatments. The techniques in this category suffer from the same problems attributed to the similarity-based methods. This includes the reliance on the parameters K and the length of the windows used for extracting the patterns.

Chandola *et al.* perform a comparative evaluation of anomaly detection techniques on sequence data [20]. They divide the previously proposed techniques into three main categories: *Kernel-based*, *Windows-based*, and *Markovian*. The Kernel-based category includes the Nearest Neighbour-based technique, which was discussed in Section 2.1.4. In the Window-based techniques (*e.g.* t-STIDE [112]), a normal profile is created from a dataset of normal sequences by extracting all windows of a fixed length w. This normal profile captures the expected frequencies of subsequences in the training data. In the test phase, all subsequences of length w are extracted from each test sequence and a 'likelihood score' is assigned to each window based on its expected frequency in the training data. The Markovian techniques, which were discussed in Section 2.1.2, and are used in this comparative study are chosen from different models including Hidden Markov Model (HMM) [112], Prob-

abilistic Suffix Tree (PST) [103], RIPPER [65], Finite State Automaton (FSA) [77], and FSAz (a variant of FSA [20]). The methods in this category often use parameters such as length, probability threshold and K, which may not be easy to set. For instance, Chandola reports that the value of the length parameter is critical in the performance of the methods used in their comparative study [20], and selecting very low or very high values for the length results in poor performances. Moreover, all the methods used in this study consider a definition of a pattern with exact matching, which means that all occurrences of a pattern in the data sequence must exactly match with the subsequence represented by the pattern.

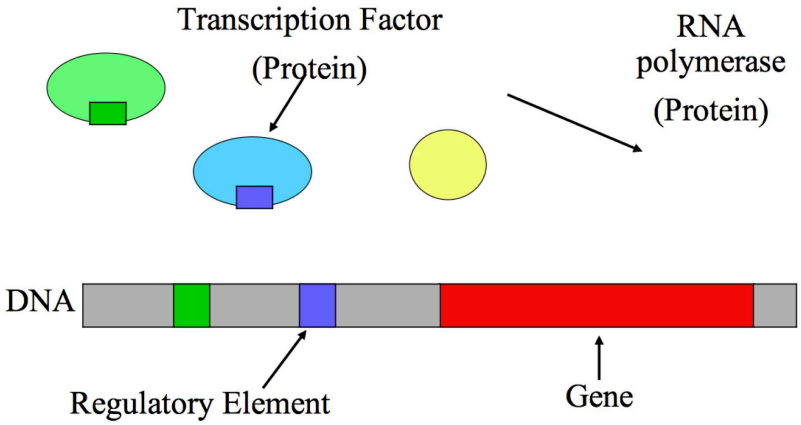
2.2 Biological Motif Discovery

The problem of *motif discovery* in the field of Bioinformatics is a close research topic to our work. Based on the hypothesis behind the motif discovery methods, and our definitions of *unexpectedly frequent* patterns, the core challenge in both problems is to find subsequences in the data which are over-represented with respect to their *expected* frequencies.

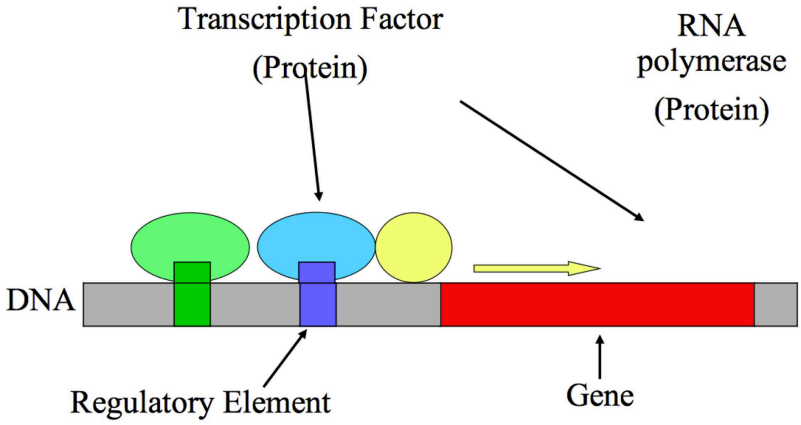
A *motif* in the field of the Bioinformatics refers to a nucleotide or amino-acid subsequence that is widespread and has a biological significance, such as being DNA *binding sites* for a regulatory protein, *i.e.*, a *transcription factor*. Transcription factors are proteins that bind to specific DNA sequences and regulate the *gene expression*. The *Gene expression* is the process by which information from a gene is used in the synthesis of a functional gene product, such as proteins. *Regulation* of gene expression refers to the control of the amount and timing of appearance of the functional product of a gene. Control of expression is vital to allow a cell to produce the gene products it needs when it needs them. The gene is the basic unit of inheritance in DNA, and is defined as a template for the copying process called *transcription*. Every gene contains a *regulatory region* typically stretching 100-1000 bp (base pair) upstream of the transcriptional start site.

The main elements involved in the gene expression regulation and two main steps of the regulation are shown in Figures 2.1(a), 2.1(b), 2.1(c). The gene expression is initiated when a transcription factor is bound to a regulatory element (*i.e.* binding site). This makes it possible for an *RNA polymerase protein* to bind to the *promoter* of the DNA sequence. The *promoter* is a special DNA sequence in the beginning of the gene to be expressed. The RNA polymerase protein is an enzyme which produces primary transcript RNA, and is necessary for constructing RNA chains using DNA genes as templates. The RNA polymerase transcribes *mRNA* (*messenger RNA*) from the DNA template. The resulting *mRNA* is a single-stranded copy of the gene, which is next translated into a new protein molecule in the *translation* step. The purpose of *translation* is to synthesize *proteins*, which are the final products and are used for for millions of cellular functions.

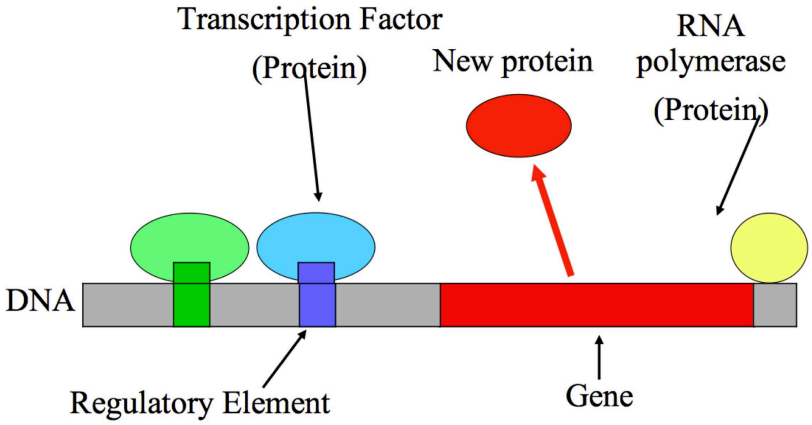
The input of a motif discovery algorithm is a collection of regulatory regions of DNA sequences that are believed to contain motifs. Core to every motif finding method is the model that is used for representing motifs. Due to large variations in binding sites of a single transcription factor, representing them is a challenging and important task. If a motif model is too specific (*e.g.* an exact string in the most extreme case), it cannot capture the variations of the binding sites. If a motif model



(a) Main elements involved in the gene expression.



(b) Gene Expression Step 1: Binding of transcription factor to the regulatory element.



(c) Gene Expression Step 2: Transcription.

Figure 2.1: Steps of Gene Regulation (adapted from [108]).

is too general on the other hand, it might match with too non-regulatory elements leading to many *false positives*.

Three types of models have been already used for motif representation:

- **Mismatch String**: is a tuple $< s, d >$, where s is a string in $\Sigma^* = \{A, C, G, T\}$, and d represents the maximum number of mismatches allowed for a binding site, with respect to the *reference* motif, to be considered a *hit* (match).
- **Position Weight Matrix (PWM)**: a PWM is a tuple $< M, t >$, in which M is a $4 \times n$ matrix, where n is the length of the motif, and each column of the matrix represents the probability distribution of the nucleotide vector $[A, C, G, T]$. A candidate string of length n is considered to be a *hit* if the sum of probabilities in log scale over respective rows are greater than the threshold t.
- **Consensus**: is a string s of length n where each position is a non-empty subset of $\{A, C, G, T\}$. These subsets correspond to the IUPAC (International Union of Pure and Applied Chemistry) symbols for DNA sequences [2] (*e.g.* W stands for $\{A, T\}$). A candidate string c is said to be a *hit* (match) against s if every position of c is a subset of respective positions in s. Otherwise, it is a not a hit.

Based on the model that is used for motif representation, two main groups of algorithms can be employed for finding the parameters of the model. Pattern-based (string-based) methods are applicable on the Mismatch String or Consensus models. This group of methods mostly rely on exhaustive enumeration of candidate patterns of different lengths and evaluating the statistical significance of patterns based on measures such as *p-value*, *z-score*, or *information content*. Probabilistic algorithms on the other hand work with Position Weight Matrices and estimate the model parameters using *Maximum-likelihood* principle or *Bayesian* inference. Once the parameters of a motif model are estimated by a motif finding algorithm, the binding sites can be reported as all strings in the input sequences which *match* a motif.

Many computational tools have been proposed for finding motifs of a specific length that are statistically overrepresented with respect to the background distribution of DNA sequences. Examples include AlignACE [55], ANN-Spec [117], Consensus [51], GLAM [38], Improbizer [5], MEME [8], MITRA [31], MotifSampler [105], oligio/dyad-analysis [110], QuickScore [88], SeSiMCMC [33], Weeder [85], YMF [97], and Seeder [32]. These tools differ in several key aspects such as in their model for representing motifs (*e.g.* Consensus [85], Position Weight Matrix [8, 33, 117], IUPAC [31], or Mismatch model [85]), and the way the statistical significance of a motif is measured (*e.g.* p-value [97], E-value [8], information content [51]), and the algorithms used to find statistically overrepresented motifs (*e.g.* Expectation-Maximization [5, 8], exhaustive search [85, 88, 97], Gibbs sampling [38, 55, 105, 117]).

From the algorithmic point of view, the computational methods for motif discovery can be divided into three main categories:

- Pattern-based algorithms: are based on (exhaustively) enumerating of candidate motifs and evaluating the statistical significance of the these motifs based on a *significance score*.

- Probabilistic algorithms: are based on a probabilistic representation of motifs in which the model parameters are estimated using the *maximum likelihood* principle or the *Bayesian* inference.
- Machine learning algorithms: use *genetics* algorithms for searching the space of motifs

In the following, we review the proposed methods in each category, and compare them with our proposed research work in the end.

2.2.1 Pattern-based Algorithms

The patten-based search methods guarantee global optimality and are appropriate for short motif. They are useful for motif finding in genomes where motifs are generally short (*e.g. eukaryotic*). Implementations based on the data structures such as *suffix trees* can make pattern-based methods fast and therefore these methods are good choices for finding totally constrained motifs. However, for typical transcription factor motifs that often have several weakly constrained positions, pattern-based methods can be problematic and the results often need to be post-processed. Another characteristic of Pattern-based methods is that they generally produce too many *spurious* (correlated) motifs. This is due to the fact motifs are evaluated individually, and a significant number of other patterns which have overlaps with a 'true' motif might appear in the result set which need to be post-processed

Tompa *et al.* [107] propose a method for enumerating motifs of specific length k, while allowing a small and a fixed number c of mismatches, and evaluating the *significance* of a motif with a given observed frequency t in the test sequences. The significance test is based on how likely it is to have t occurrences of a motif if the background sequences are generated based on a *random* or a *Markov chain* distribution. The proposed score in this work is the *z-score*, which measures the number of standard deviations by which the observed frequency value t exceeds its *expected* value.

Sinha *et al.* [97] extend the motif model to add support for gaps (*spacers*) between conserved parts of the motifs, and propose the *YMF* algorithm which uses a similar approach for evaluating the statistical significance of motifs based on the *z-score* measure. The inputs to the YMF algorithm are a set of upstream sequences, the number of non-spacer characters in the motifs to be enumerated, and the transition matrix for a Markov chain model of order m (which is constructed from the regulatory regions of genome in *yeast*). The YMF algorithm then selects motifs with greatest *z-scores*.

The *Weeder* algorithm [85] enumerates the motifs of different lengths (*e.g.* 6, 8, 10, 12), allowing a small percentage of the sequence for mismatches, and calculates a *significance score* for each motif which roughly compares the observed frequencies of motifs with their 'expected' frequencies. A suffix tree data structure is used to speed-up pattern enumeration. The MITRA (Mismatch Tree Algorithm) [31] proposed by Eskin *et al.* also uses the *suffix tree* data structure for motif enumeration. Other graph-theoretic data structures are combined with pattern-based algorithms such as WINNOWER [87] and cWINNOER [68].

2.2.2 Probabilistic Algorithms

In this category of approaches, the motifs are modeled by probabilistic models, mainly the well-known *position weight matrix* (*PWM*). A PWM has one row for each symbol of the alphabet (*e.g.* 4 rows for nucleotides in DNA sequences). A basic *PWM* using relative frequencies is constructed by counting the occurrences of each symbol at each position and then normalizing the values across each column. The main assumption behind the probabilistic algorithms is that sequences follow a background distribution (*e.g.* modelled by a *Markov chain* for instance) except at positions of transcription factor binding sites, in which the sequence contains an instance of a motif following another distribution which is modelled by a probabilistic *PWM*.

Probabilistic methods have the advantage of requiring few search parameters but rely on probabilistic models of the regulatory regions, which can be very sensitive with respect to small changes in the input data. The algorithms developed based on the probabilistic models have been used to find longer or more general motifs than are required for transcription factor binding sites. Therefore, these methods are more appropriate for motif discovery in species where the motifs are generally longer (*e.g. prokaryotes*). However, these algorithms are not guaranteed to find globally optimal solutions, since they are based on some form of local search methods, such as *Gibbs sampling*, *Expectation-Maximization* (*EM*) or greedy algorithms that may converge to a locally optimal solution.

Hertz *et al.* use a *greedy* algorithm based on a probabilistic model to find the *binding site* with the highest *information content*. The assumption in their algorithm was that there is a common motif in every sequence. This algorithm has been substantially improved over the years. Notably, in their latest algorithm (*Consensus*), Hertz and Stormo [51] propose a method to iteratively find motifs with highest *information contents* without the initial assumption of existence of a motif in each input sequence.

Most of the proposed probabilistic methods use *Expectation-Maximization* (*EM*) or *Gibbs-sampling* for estimating model parameters. The *EM* algorithm was first used by Lawrence *et al.* [64] based on the assumption that each sequence must contain at least one binding site. This assumption was relaxed in a later work by Bailey *et al.* [7, 8]. Moreover, subsequences that actually occur in the *biopolymer* sequences were used as *starting points* for *PWM*s to increase the chance of finding globally optimum motifs.

The *Gibbs sampling* is another widely used method for estimating the parameters of the model in probabilistic approaches. The Gibbs sampler is a *Markov Chain Monte Carlo* (*MCMC*) approach. In the basic Gibbs sampling technique proposed by Lawrence *et al.* [63], it is assumed that we are given a set of N sequences S_1,...,S_N and we seek within each sequence mutually similar segments of specified length ℓ. The algorithm maintains two evolving data structures. The first one is the pattern model (*e.g.* in the form of a *PWM*), and the second one is the *alignment* model, constituting the positions of patterns in the sequences. The algorithm starts by choosing random starting positions within the various sequences, and goes through many iterations of "Predictive update" and "Sampling" steps to locate the *alignment* that maximizes the ratio of the corresponding *pattern* probability to *background* probability.

Other variations of the basic Gibbs sampling algorithm have been proposed in the literature. The *AlignACE* (Aligns Nucleic Acid Conserved Elements) algorithm [55] is one such a method that is different from the basic Gibbs sampling algorithm in using an improved near-optimum sampling method, and using the *MAP* (maximum a priori log-likelihood) score to judge different motifs sampled. The *MAP* is a measure of the degree of overrepresentation of a motif as compared to the expected random occurrence of that motif in the sequence under consideration. The *MotifSampler* algorithm developed by Thijs *et al.* [105], extends the Gibbs sampling algorithm to support background distributions with higher-order Markov chain models. The BioProspector algorithm [72], developed by Liu *et al.*, uses a Gibbs sampling strategy, but allows for the modeling of spaced motifs and motifs with *palindromic* patterns, and also provides support for Markov models of order 0 to 3. The GibbsST algorithm utilizes the Simulated tempering technique (from the field of thermodynamics) to reduces the vulnerability of the basic Gibbs sampling algorithm to local optima [96].

2.2.3 Machine Learning Algorithms

Liu et al. [53] apply *genetic algorithms (GA*s) to the problem of motif discovery. In their proposed algorithm *FMGA*, the *mutation* and *cross-over* operators are defined to reserve the completely conserved positions in position weight matrices, as well as the specially designed *gap penalties* to produce the optimal child patterns. Rearrangements in a *PWM* is also performed to avoid local optima. A *self-organizing neural network* is proposed by *Liu et al.* for motif discovery in DNA and protein sequences [71]. The neural network contains several layers, with each layer performing classifications at different levels. The top layer divides the input space into a small number of *regions*, and the bottom layer classifies all input patterns into *motifs* and *non-motif* patterns. The network will grow as needed, and depending on the number of input patterns to be classified, several layers between the top layer and the bottom layer might be added to perform intermediate classifications.

Hu et al. [58] employ an *ensemble* approach for motif finding to improve the prediction power of the motif finding algorithms. The main idea of an Ensemble approach is to take advantage of superb predictions of every component algorithm, while hoping that the weak predictions of an algorithm can be covered by predictions made by other algorithms. Hu *et al.* propose a novel clustering-based ensemble algorithm named *EMD* for motif discovery by combining multiple predictions from multiple runs of one or more arbitrary motif finding algorithms. The authors use five component algorithms namely AlignACE [55], Bioprospector [72], MDScan [73], MEME [8] and MotifSampler [105] in their study.

Analysis and Comparison to Our Research: As it was discussed, the main challenge in the motif finding problem and the problem addressed in our research is to find subsequences in the data which are over-represented with respect to an expected distribution. However, our work is different from the proposed motif finding methods in several aspects. While these computational tools have been developed particularly for finding *motifs* in Biological sequences, our work presents a general framework for finding *surprising* patterns in sequence data (not limited to Bioinformatics data). Also, the motif discovery methods use biology-motivated heuristics for finding the length of true binding sites from a large number of statistically sig-

nificant patterns. Researchers in this domain [85] have verified that the patterns with highest scores do not necessarily correspond to true binding sites. In contrast, we mathematically formalize the problem of finding *core* surprising patterns from a large number of significant patterns and present a greedy algorithm for finding these *core* patterns among all other redundant patterns.

2.3 Non-Redundant Subspace Clustering in High Dimensional Data

Our research work also shares some challenges with the *projected* and *subspace* clustering in high-dimensional data. *Projected clustering* computes several disjoint clusters so that each cluster exists in its own subset of attributes. Projected clustering seeks to assign each point to a unique cluster, but clusters may exist in different subspaces. *Subspace clustering* enumerates clusters of points in all subsets of attributes. This means that a point might be a member of multiple clusters, each existing in a different subspace. One of the challenges of clustering in the high-dimensional data is that lots of overlapping clusters might be created by projected or subspace clustering methods. Identifying the 'true' clusters among a large set of overlapping clusters is not an easy task. Moreover, the number of possible subspace projections is exponential in the number of dimensions, and this makes the task of finding relevant clusters more challenging. A long sequence is an example of a high-dimensional dataset, and the same issue of *redundancy* is encountered by methods seeking to find statistically significant subsequences in the data without knowing their lengths. The redundancy occurs, as it was discussed shortly and will be demonstrated through an experimental study in Section 3.5, due to the fact that the subsequences which partially overlap with *true* statistically significant patterns are likely to be rendered as significant patterns in the result set.

Some efforts have been made by researchers to mitigate the issues of redundancy and improve the relevance of the clustering results. In their proposed *Density-Unbiased Subspace Clustering* (*DUSC*) model, Assent *et al.* [6] introduce the notion of redundant subspace clusters, as a cluster which is repeated in a higher dimensional subspace cluster. They argue that users are generally only interested in seeing a lower dimensional subspace cluster if the number of new objects in this lower dimensional projection is sufficiently large, whereas more dimensions describe a more specific and thereby informative pattern than the trivial patterns in only few dimensions. Using the *DUSC* model, Assent *et al.* propose a new algorithm (*INSCY*: *indexing subspace clusters with in-process-removal of redundancy*) for removing *repeated* (redundant) clusters. The issue of large number of possible clusters is addressed by adopting a *depth-first* search strategy, so that high-dimensional subspace clusters are detected first.

Muller *et al.* introduce a new *global relevance* model, called *RESCU*, for subspace clustering [82]. Their goal is to find *all interesting* clusters, and only *non-redundant* ones. In this work, a new *interestingness* function for subspace clusters is combined with a *coverage* criterion for an overall redundancy removal. By reducing from the *Set Cover* problem, it is shown that the defined *relevant subspace clustering* problem is NP-hard, and an approximation algorithm is proposed to generate cluster candidates in the *best-first* order according to their *relevance*.

Moise *et al.* [79] formulate the problem of finding non-redundant clusters in high-dimensional data. Assuming a uniform distribution for data, they propose an algorithm for finding the minimum number of statistically significant axis-parallel *regions*. Intuitively, a statistically significant *region* is a region that contains significantly more points than expected.

Analysis and Comparison to Our Research: Our work is similar to the problems of high-dimensional subspace and projected clustering in the sense that we have to deal with high-dimensional data and address the issue of *redundancy* in the result set, which is encountered similarly in clusters in different subspaces. However, finding statistically significant patterns in sequences are different from high-dimensional clustering in several aspects. The main difference between our problem and the works in this category is that the temporal relation which is inherent in every sequence data does not exist in subspace clusters. In order to capture the temporal and ordered relation between data points, different models (such as Markov chain models) should be employed. A different underlying model changes the concepts of statistically significant subsequences, and the correlations between patterns in our problem with those defined in the subspace clustering. Also, the models required to capture the concept of approximate matches in sequences are very different from those in the subspace clustering. As a result, finding statistically significant patterns in long sequence data introduce new challenges that have not been addressed already in the domain of subspace clustering.

2.4 Motif Discovery in Time Series Data

Another line of research related to our work is 'motif' discovery in time series data. Motifs are defined as similar subsequences in time series data that are observed frequently. The motifs, as defined in this lines of work, are similar to the unexpectedly frequent patterns that we intended to find in our research work.

This problem was first formalized by Patel *et al.* [84] and an algorithm was proposed for finding subsequences of fixed length with highest counts of neighbours. A new type of motifs that is invariant to *uniform scaling* is proposed by Yankov *et al.* [122]. Chiu *et al.* propose a probabilistic approach to finding motifs of a given length in time series data [22]. Castro *et al.* propose a method for evaluating the significance of subsequences of given lengths using statistical tests [16]. These works are different from our work in the definition of patterns, the type of the data, and also the required input parameters.

Some works have addressed the problem of motif discovery in time series without knowing the lengths of the subsequences to be discovered [34,67]. Ferreira *et al.* introduce approximate motifs, as clusters that contain a minimum number of time series subsequences with a minimum similarity [34]. Li *et al.* propose a method for identifying approximate variable-length time series motifs [67]. Their approach is based on a grammar-based compression algorithm that can discover frequent patterns in the data. The definitions of patterns in these works are either based on some user-defined parameters (*e.g.* minimum count, minimum similarity, *etc.*) or represent a summarized view of the data, which makes them different from statistically significant patterns in our work.

Analysis and Comparison to Our Research: While the works in this cat-

egory also intend to find frequent subsequences in time series sequences, they are different from our research in several aspects, including the notion of interesting patterns, the nature of the input data, and the input parameters.

2.5 Mining Frequent Patterns in Sequence Data

Mining unexpectedly frequent patterns in sequence data is another close research topic to our research study. Numerous works have been published around this topic addressing different aspects such as what measure is used to define *unexpectedness*, what defines a pattern, how a data sequence is defined, and how the performance of a method is evaluated.

Dutta *et al.* propose a method for mining *statistically significant* patterns in a long sequence dataset [100]. Their notion of significant patterns is based on the *Chi-square* statistic (χ^2) and two heuristics are developed for returning the top-k substrings with the largest χ^2 measure. To improve upon the naive algorithm which computes the chi-square for each substring individually, the *local maxima* of a string is introduced. The *local maxima* is defined to be a substring such that while traversing through it, the inclusion of the next symbol does not decrease the χ^2 value of the resultant substring. Two algorithms, *All-Pair Refined Local Maxima Search* ($ARLM$) and *Approximate Greedy Maximum Maxima Search* ($AGMM$), are proposed based on the notion of a *local maxima* to efficiently search and identify statistically significant patterns within a long sequence. The definition of patterns in this work is again limited to simple string representations (*i.e.* no *mismatch* allowed in the definition of a pattern) which makes this method incapable of finding approximate patterns. Moreover, this work does not address the issue of selecting the *length* of true deviating (*i.e.* statistically significant) patterns, which can lead to generating highly correlated patterns in the top-k solution and possibly missing some of the 'true' significant patterns.

A more *generalized* notion of patterns has been used in some research works, in which the patterns represent an *approximate* set of subsequences in sequence data sets. Floratou *et al.* [35] have proposed *FLAME* as a motif mining framework in sequence datasets. The proposed *motif model* in this work depends on four parameters (L, M, s, k), where L denotes the *length* of the pattern, M denotes the *distance matrix* (which is used to compute the similarity between a given string and the reference motif), s denotes the maximum distance threshold, and k denotes the minimum support required for a pattern to qualify as a *motif*. The proposed motif model is supposed to represent the set of strings of length L with minimum frequency of k whose distance to the reference motif with respect to the distance matrix M is less than the threshold s. To improve upon the naive algorithm (e.g. exploring all possible motifs and computing the *support* for each of them), two *suffix trees* are used to guide the search strategy in a more efficient way. The first tree is based on the input data sequences, which is referred to as the data suffix tree, and the second one is based on all possible motif strings, which is referred to as the *model suffix tree*. The computed supports on the *data suffix tree* are used to prune branches of the *model suffix tree* which are guaranteed not to produce any results under the model. The model proposed in this work depends on some input parameters (*e.g.* distance matrix M, minimum support k) which are domain-dependent, and cannot

be determined intuitively. Moreover, the evaluation of the method has been focused on the run-time performance rather than the matching extent to the 'true' motifs in the data, which is the focus of our work.

In another work in this category, Zhu *et al.* propose a method for finding *approximate* sequential patterns, in which variations between patterns is represented by the *Hamming distance* [4]. The motif model presented in this work as well depends on two input parameters *minimum support* and *maximum distance* which affect the robustness of the method. The main focus of this work is to enumerate the motifs and compute their *supports* efficiently. The proposed search strategy consists of two steps *break-down* and *build-up*. The *break-down* step is based on the observation that all occurrences of a frequent pattern can be classified into groups, which are called *strands*. The strands are mined out by an iterative growth. In the *build-up* step, these strands are combined together to form the support sets from which all approximate patterns can be identified. Similar to *FLAME*, the proposed work by Zhu depends on some non-intuitive input parameters such as *support*, and focuses on improving the running time rather than accuracy in terms of matching with the ground truth.

Yang *et al.* propose a method for finding surprising patterns in sequence data [121]. Their notion of surprise is modelled based on the *information gain* and they allow some 'don't care' positions in the pattern specification. Even though the proposed notion of surprise in this work allows them to compare the significance of patterns of different lengths, the type of patterns discovered by this method is different from those in our work. The main focus of Yang's work is on finding patterns which occur frequently within a period (i.e. *repetitive patterns*) in a sequence. Our work, on the other hand, does not impose any constraint on distances between occurrences of a pattern. The work proposed by Keogh *et al.* [59] for finding surprising patterns in a time series database is similar to ours in that the discovered patterns should be observed more frequently than expected. However, the score used in this work for measuring the extent of *surprise* of patterns is based on the raw counts of observed patterns compared with their counts in the training data, and using a user-provided threshold for choosing the surprising pattern. In our proposed method, we estimate the p-values of patterns, which gives a more robust means to choose the patterns based on a significance level. Also, the patterns in Keogh's work [59] are extracted based on different window lengths, and no attempt is made in choosing the right length of discovered patterns.

Sequential pattern mining is a close research area to finding statistically significant patterns in sequence data. This topic has been addressed in numerous publications, including the seminal work by Agrawal *et al.* [4] and the improvements proposed over Agrawal's work in algorithms such as SPADE [124] and BIDE [111]. The primary focus in this line of research is on mining a sequence of symbols with arbitrary gaps between them, whereas our work is focused on finding contiguous patterns. Some algorithms such as cSPADE [123], CloSpan [120], PrefixSpan [86], Gap-BIDE [66], and Gap-Connect [66] allow certain constraints on the maximum gap between two consecutive symbols, and as such can be adapted to mine for contiguous subsequences, as defined in our work. However, the applications of these algorithms are limited to finding exactly matching subsequences due to the fact that no notion of noise or approximation is allowed in the pattern definition.

Gwadera *et al.* address the problem of finding significant episodes in an event sequence [47], where the definition of an episode is limited to subsequences occurring in a time window of fixed size. In another work, Tatti *et al.* address the problem of summarizing a data sequence with the "best" set of serial episodes based on the MDL principle [104]. There is no notion of the deviation from a model in these types of patterns, making them different from statistically significant patterns in our work. Webb *et al.* propose a method for finding statistically significant association rules assuming a database of transactions [113].

Gwadera *et al.* propose a method for evaluating and ranking the significance of sequential patterns (itemset-sequences) compared to a reference model [48]. In another similar work [74], Low-Kam *et al.* propose a method for finding statistically significant *sequential patterns* to reduce the set of discovered patterns. They introduce the new measure of interestingness of sequential patterns based on their frequency under the *null model*, and also use the notion of *unexpected closed* sequential patterns to limit the set discovered patterns to the most interesting patterns. In another proposed work for finding surprising sequential patterns, Chakrabarti *et al.* propose a method for finding unexpected sequential patterns in market basket data [17]. The notion of interestingness used in this work is derived from the *minimum description length* principle, and replaces the domain knowledge parameters such as support, confidence, or window length. However, these methods are proposed for finding sequential patterns in sequences of itemsets (*e.g.* transaction-based databases), rather than the contiguous sequence data model assumed in our work.

Chapter 3

Background and Problem Statement

3.1 Markov Chain Model

A *Markov chain* model is a discrete stochastic process with the *Markovian* property. More formally, a Markov chain is a sequence of random variables X_1, X_2, X_3, ... with the property that, given the *present* state, the *future* and *past* states are independent:

$$P(X_{n+1} = x|X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = P(X_{n+1} = x|X_n = x_n) \quad (3.1)$$

Note that we are not saying that, for example X_{10} and X_1 are independent. They are not. However, given X_9, X_{10} is conditionally independent of X_1. This specific kind of "*memorylessness*" is called the Markov property. A Markov chain is *stationary* if at any time, it has the same transition probabilities.

The changes of state of the system are called *transitions*, and the probabilities associated with various state changes are called *transition probabilities*. A Makov chain model is characterized by three parameters *state space*, a *transition matrix*, describing the probabilities of transitions, and the distribution of *initial state*.

A more general class of Markov chains adds a new parameter m to the model stating the depth in the history of the states the next state depends on. A Markov chain of order m (or a Markov chain with memory m) is a process in which the *future* state depends on the past m states:

$$P(X_{n+1} = x|X_1 = x_1, ..., X_n = x_n) = P(X_{n+1} = x|X_n = x_n, ..., X_{n-m+1} = x_{n-m+1}) \quad (3.2)$$

Markov chains have many applications as statistical models of real-world processes such as stock prices, web page navigations, and probabilistic moves in games. In many real-world applications, the distribution of the process which generates the data are unknown, and we just approximate this distribution using a Markov model. Assuming a Markov chain of order m as the underlying model for a process, the probability of a sequence $S = x_1x_2...x_n$ of length n under this model is calculated as follows:

$$P(S) = P(x_1)P(x_2|x_1)P(x_3|x_1x_2)...P(x_n|x_{n-m}x_{n-m+1}...x_{n-1}) \tag{3.3}$$

The initial probability $P(x_1)$ and the following conditional probabilities are calculated from the model parameters, including the initial distribution of states and the transition matrix, correspondingly. In practice, the model parameters are derived from a *reference* data (i.e. training data) which represents the normal behaviour of the process or should be provided by the user. In the following chapters, we are using a stationary Markov chain as the underlying model for sequence data.

3.2 Smoothing for Markov Models

Accuracy of probabilities in the Markov Chain model depends on the training data available for learning the model. Achieving accurate probability estimation is challenging in cases where sufficient data is not available or a higher-order Markov model should be learned. To address this issue, we are using the *smoothing* techniques that are originally used in the domain of Language Modelling. The dominant technology in language modelling is *n-gram* models, and *smoothing* methods are used to better estimate probabilities when there is insufficient data to estimate probabilities accurately [21].

A language model is usually formulated as a probability distribution $p(s)$ over strings s that attempts to reflect how frequently a string s occurs as a sentence. The most widely-used language models, by far, are n-grams. For a sequence s composed of the symbols (or 'words' which are building units of language models) $w_1, ..., w_l$, we can express $p(s)$ as

$$p(s) = p(w_1)p(w_2|w_1)p(w_3|w_1w_2)...p(w_l|w_1...w_{l-1}) = \prod_{i=1}^{l} p(w_i|w_1...w_{i-1}) \tag{3.4}$$

As an example, for $n = 2$ these models are called *bigrams*. This corresponds to a Markov Chain of order $m = 1$ (in general, an n-gram model can be interpreted as a Modev model of order $n-1$), in which we make the assumption that the probability of each symbol depends only on the preceding symbol, giving us

$$p(s) = \prod_{i=1}^{l} p(w_i|w_1...w_{l-1}) = \prod_{i=1}^{l} p(w_i|w_{i-1}) \tag{3.5}$$

To make $p(w_i|w_{i-1})$ meaningful for $i = 1$, we can pad the beginning of the sentence with a distinguished token <BOS>; that is, we pretend w_0 is <BOS>, as proposed in [21]. To estimate $p(w_i|w_{i-1})$, the frequency with which the word w_i occurs given that the last word is w_{i-1}, we can simply count how often the bigram $w_{i-1}w_i$ occurs in some text. Let $c(w_{i-1}w_i)$ denote the number of times the bigram $w_{i-1}w_i$ occurs in the given text. Then, we can calculate the conditional probability as follows:

$$p(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i)}{\sum_{w_i \in \Sigma} c(w_{i-1}w_i)} \tag{3.6}$$

where Σ represents the alphabet of symbols. A *training data* is required for learning model parameters. The estimate for $p(w_i|w_{i-1})$ given in equation 3.6 is called the

maximum likelihood (ML) estimate of $p(w_i|w_{i-1})$ because this assignment of probabilities yields the bigram model that assigns the highest probability to the training data of all possible bigram models. Estimation of conditional probabilities based on ML models highly depends on the training data. This means that some probabilities might be evaluated to 0 just because a specific sequence of symbols (or 'words') are not seen in the training data.

A *smoothing* technique intends to address this problem. The name *smoothing* comes from the fact that these techniques tend to make distributions more uniform, by adjusting low probabilities such as zero probabilities upward, and high probabilities downward. *Smoothing* refers to methods that assign probabilities to events (N-grams) that do not occur in the training data. According to a pure maximum-likelihood estimator these events would have probability zero, which is plainly wrong since previously unseen events in general do occur in independent test data. Because the probability mass is redistributed away from the seen events toward the unseen events the resulting model is "smoother" (closer to uniform) than the ML model. Discounting refers to the approach used by many smoothing methods of adjusting the empirical counts of seen events downwards.

3.2.1 Additive Smoothing

Additive smoothing is the simplest type of smoothing used in practice. The simple idea behind this technique is that the frequency of each n-gram is pretended to be a constant value δ more that it is actually observed in the training data, where typically $0 < \delta \leq 1$, *i.e.*,

$$p_{add}(w_i|w_{i-n+1}^{i-1}) = \frac{\delta + c(w_{i-n+1}^{i})}{\delta|V| + \sum_{w_i} c(w_{i-n+1}^{i})} \tag{3.7}$$

where V is the vocabulary, the set of all symbols being considered, and the notation w_j^i $(i < j)$ represents the substring starting from index i to index j (*i.e.* $w_j w_{j+1} ... w_{i-1} w_i$).

Even though the *additive* smoothing provides a simple fix to the problem of unseen n-grams, it comes with a cost in performance. Gale and Church have argued that in order for *additive* smoothing to work the frequencies of frequencies of n-grams should follow a *geometric law* (linear on semi-log), an assumption that is violated quite often in practice [40,41]. As a result, the additive smoothing generally performs poorly, and it can even produce worse results than the Maximum-likelihood estimate.

3.2.2 Good-Turing Estimate

The Good-Turing estimate [46] is central to many smoothing techniques. The main idea behind this estimate is that that for any n-gram that occurs r times, we assume that it occurs r^* times, where

$$r^* = (r+1)\frac{n_{r+1}}{n_r} \tag{3.8}$$

such that r_n is the number of n-grams that occur exactly r times in the training data (*i.e.* frequencies of frequencies). To derive a probability value from this adjusted

count, we normalize using the total counts of n-grams. More formally, for an n-gram α which is observed with frequency r, the Good-Turing estimate is calculated as follows:

$$p_{GT} = \frac{r^*}{N} \tag{3.9}$$

where N is the total number of frequencies of frequencies, *i.e.* $N = \sum_{r=0}^{\infty} n_r r^*$.

In practice, the Good-Turing estimate is not used by itself for n-gram smoothing, because it does not include the combination of higher-order models with lower-order models necessary for good performance, as discussed in the following sections. However, it is used as a tool in several smoothing techniques.

3.2.3 Katz Estimate

Katz smoothing (1987) extends the intuitions of the Good-Turing estimate by adding the combination of higher-order models with lower-order models [57]. The *katz* smoothing works by calculating the *adjusted* counts of n-grams. As an example, for a bigram w_{i-1}^i with count $r = c(w_{i-1}^i)$, its *adjusted* count is calculated using the following formula:

$$c_{katz}(w_{i-1}^i) = \begin{cases} d_r r & \text{if } r > 0 \\ \alpha(w_{i-1}) p_{ML}(w_i) & \text{if } r = 0 \end{cases} \tag{3.10}$$

This means that all bigrams with a nonzero count r are *discounted* according to a discount ratio d_r and the discounted value is distributed among the zero-count bigrams. The discount ratio d_r is approximately $\frac{r^*}{r}$, the discount predicted by the Good-Turing estimate. The counts subtracted from the nonzero counts are then distributed among the zero-count bigrams according to the next lower-order distribution, i.e., the unigram model in the case of estimating the counts of bigrams. The value $\alpha_{w_{i-1}}$ is chosen so that the total number of counts in the distribution $\sum_{w_i} c_{katz}(w_{i-1}^i)$ is unchanged; *i.e.* $\sum_{w_i} c_{katz}(w_{i-1}^i) = \sum_{w_i} c(w_{i-1}^i)$. Also, the sum of probabilities of all bigrams starting with w_{i-1} ($\sum_{w_i} p_{katz}(w_i|w_{i-1})$) should equal to 1, and considering the fact that part of probability mass is discounted from the seen bigrams and added to the unseen bigrams with factor $\alpha_{w_{i-1}}$, the value $\alpha_{w_{i-1}}$ is derived as follows:

$$\alpha_{w_{i-1}} = \frac{1 - \sum_{w_i : c(w_{i-1}^i) > 0} p_{katz}(w_i|w_{i-1})}{\sum_{w_i : c(w_{i-1}^i) = 0} p_{ML}(w_i)} = \frac{1 - \sum_{w_i : c(w_{i-1}^i) > 0} p_{katz}(w_i|w_{i-1})}{1 - \sum_{w_i : c(w_{i-1}^i) > 0} p_{ML}(w_i)} \tag{3.11}$$

In *Katz* smoothing, the large counts are believed to be reliable, and therefore, the discount ratio d_r is considered to be 1 for all $r > k$ (Katz suggests a value of 5 for k). Based on the ideas from the Good-Turing smoothing, Katz proposes the following formula for computing the discount ratios for all counts less than k:

$$d_r = \frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}} \tag{3.12}$$

From the corrected ratio d_r in equation 3.12 and the equation 3.11, the smoothed probabilities can be can be computed by normalization, as follows

$$p_{katz}(w_i|w_{i-1}) = \frac{c_{katz}(w_{i-1}^i)}{\sum_{w_i} c_{katz}(w_{i-1}^i)} \tag{3.13}$$

As we can see in equation 3.10, the bigram model is defined in terms of the unigram model; in general, the Katz n-gram model can be defined in terms of the Katz (n-1)-gram model in a similar way. To end the recursion, the Katz unigram model is taken to be the maximum likelihood model.

3.2.4 Witten-Bell Estimate

Witten-Bell smoothing was originally developed for the task of text compression [116]. The main idea behind the Witten-Bell estimate is to use the information in the lower-order n-gram model in order to estimate the probabilities in the higher-order models. More formally, the higher-order n-gram models are *interpolated* based on the lower-order n-gram models, as follows:

$$p_{WB}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}} p_{ML}(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}}) p_{WB}(w_i|w_{i-n+2}^{i-1}) \tag{3.14}$$

That is, the n^{th}-order smoothed model is defined recursively as a linear interpolation between the n^{th}-order *maximum likelihood* model (*i.e.* $p_{ML}(w_i|w_{i-n+1}^{i-1})$) and the $(n-1)^{th}$-order *smoothed* model (*i.e.* $p_{WB}(w_i|w_{i-n+2}^{i-1})$).

To motivate Witten-Bell smoothing, we can interpret equation 3.14 as saying:

- with probability $\lambda_{w_{i-n+1}^{i-1}}$ we should use the higher-order model, and
- with probability $1 - \lambda_{w_{i-n+1}^{i-1}}$ we should use the lower-order model.

The parameters $\lambda_{w_{i-n+1}^{i-1}}$ in the *Witten-Bell* smoothing are computed by counting the number of unique words that follow the history w_{i-n+1}^{i-1}. This value is denoted by $N_{1+}(w_{i-n+1}^{i-1}\bullet)$, formally defined as

$$N_{1+}(w_{i-n+1}^{i-1}\bullet) = |w_i : c(w_{i-n+1}^{i-1} w_i) > 0| \tag{3.15}$$

where N_{1+} represents the number of words with a count of one or more, and the symbol '$\bullet$' represents a free variable which can take any value in the vocabulary V. The parameters $\lambda_{w_{i-n+1}^{i-1}}$ is the probability that a symbol not observed after the history w_{i-n+1}^{i-1} in the training data occurs after that history in a test sequence. Witten *et al.* show that this probability can be estimated using the following equation:

$$1 - \lambda_{w_{i-n+1}^{i-1}} = \frac{N_{1+}(w_{i-n+1}^{i-1}\bullet)}{N_{1+}(w_{i-n+1}^{i-1}\bullet) + \sum_{w_i} c(w_{i-n+1}^{i})} \tag{3.16}$$

This gives us the following recursive formula for calculating a discounted conditional probability with *Witten-Bell* smoothing:

$$p_{WB}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i}) + N_{1+}(w_{i-n+1}^{i-1}\bullet)p_{WB}(w_i|w_{i-n+2}^{i-1})}{\sum_{w_i} c(w_{i-n+1}^{i}) + N_{1+}(w_{i-n+1}^{i-1}\bullet)} \tag{3.17}$$

To end the recursion, we can take the smoothed 1^{st}-order model to be the maximum likelihood distribution, or we can take the smoothed 0^{th}-order model to be the *uniform* distribution.

Smoothing is generally done in one of two ways. The *Interpolated* models take lower-order counts into account no matter if the query word is seen in the training data or not. However, the *backoff* models only consider lower-order counts when the query word is not seen in the training data.

The backoff version of the Witten-Bell smoothing technique is calculated as follows:

$$p_{WB}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} f(w_{i-n+1}^{i-1}w_i) & \text{if } c(w_{i-n+1}^{i-1}w_i) > 0 \\ bow(w_{i-n+1}^{i-1})p_{WB}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i-1}w_i) = 0 \end{cases} \tag{3.18}$$

where the functions $bow(.)$ and $f(.)$ represent the back-off value and the conditional probability without back-off to the lower-order model, respectively, and are defined as follows:

$$bow(w_{i-n+1}^{i-1}) = \frac{1 - \sum_{w_i:c(w_{i-n+1}^{i})>0} f(w_{i-n+1}^{i-1}w_i)}{1 - \sum_{w_i:c(w_{i-n+2}^{i})>0} f(w_{i-n+2}^{i-1}w_i)} \tag{3.19}$$

$$f(w_{i-n+1}^{i-1}w_i) = \frac{c(w_{i-n+1}^{i})}{\sum_{w_i} c(w_{i-n+1}^{i}) + N_{1+}(w_{i-n+1}^{i-1}\bullet)} \tag{3.20}$$

3.3 Statistical Significance Testing

A *statistical hypothesis test* is a method of supporting or rejecting claims based on a sample data. In statistics, a result is called statistically significant if it is determined as unlikely to have occurred by chance alone, with respect to a predetermined threshold probability, the *significance level*.

Every significance test begins with a *null hypothesis* H_0. The null hypothesis represents a theory that is either believed to be true or because it is to be used as a basis for argument, but has not been proved. For example, in a clinical trial of a new drug, the *null hypothesis* might be that the new drug is no better, on average, than the current drug. We would write H_0: "there is no difference between the two drugs on average". The alternative hypothesis H_1 is the opposite of the null hypothesis; this is usually the hypothesis that is set up to investigate. For example, in a clinical trial of a new drug, the alternative hypothesis might be that the new drug has a different effect, on average, compared to that of the current drug. It is written as H_a: "the two drugs have different effects, on average".

After establishing the *hull* hypothesis and the *alternative* hypothesis, the p-value probability is calculated. The p-value is the estimated probability of rejecting the *null hypothesis* (H_0) of a study question when that hypothesis is actually true. The

p-value is then compared to a *significance level* α. Typical values for α are 0.05, 0.01, and 0.001. If the p-value is less than the significance level α, the null hypothesis is rejected at the significance level α and the alternative hypothesis is accepted. Otherwise, the null hypothesis is accepted, which means the test has no result (*i.e.* the evidence is insufficient to support a conclusion).

In summary, the statistical significance testing consists of four steps as follows:

1. Establish the hypotheses: This involves stating the *null* and *alternative* hypotheses. The hypotheses are stated in such a way that they are *mutually exclusive*. That is, if one is true, the other must be false.

2. Compute from the observations the observed value t_{obs} of the test statistic T, and select a significance level α.

3. Calculate the *p-value*, the probability of observing a test statistic at least as extreme as that which was observed, under the null hypothesis.

4. Interpret results: reject the null hypothesis, in favour of the alternative hypothesis, if and only if the p-value is less than the significance level α threshold.

Two types of errors can result from a hypothesis test:

- *Type I error (False Positive)*: A Type I error occurs when the a null hypothesis is rejected and it is actually true. The probability of committing a Type I error is called the *significance level* (α).

- *Type II error (False Negative)*. A Type II error occurs when the researcher fails to reject a null hypothesis that is false. The probability of committing a Type II error is called *Beta*, and is often denoted by β.

3.4 Correction for Multiple Hypothesis Testing

Multiple testing refers to any instance that involves the simultaneous testing of several hypotheses. In a statistical test, the significance level α represents the probability of the null-hypothesis being rejected when in fact it is true (*i.e.* the probability of a "false positive" or *Type I* error) for a single statistical test. However, in many applications the significance of multiple observations should be compared against a null hypothesis. When performing multiple tests simultaneously, the probability of false positives is not equal to α. If one does not take the multiplicity of tests into account, then the probability that some of the true null hypotheses are rejected by chance alone may be large.

Example 2 *Consider a case where* 20 *hypotheses have to be tested, and a significant level is set at* 0.05. *Given this, and assuming independence of the tests, the probability of observing at least one significant result just due to chance is calculated as follows:*

$$\begin{aligned}\Pr(\textit{at least one significant result}) &= 1 - \Pr(\textit{no significant results}) \\ &= 1 - (1 - 0.05)^{20} \\ &\approx 0.64\end{aligned}$$

So, with 20 tests being considered, we have a 64% chance of observing at least one significant result, even if all of the tests are actually not significant. In genomics and other biology-related fields, it's not unusual for the number of simultaneous tests to be much larger than 20, and the probability of getting a significant result simply due to chance keeps going up. In statistics, this problem is referred to as *multiple hypothesis testing* [78].

Methods for dealing with multiple testing are usually based on adjusting the significance level in some way, so that the probability of observing at least one significant result due to chance remains below your desired significance level.

3.4.1 Classic Benferroni Method

A definition of statistical significance for multiple hypothesis tests involves the probability of making one or more Type I errors among the family of hypothesis tests, called the *family-wise error rate* (FWER). The classic Benferroni correction is a an example of the *FWER* class of methods. The Benferroni method adjusts the significance level α for a single test by dividing it by the number of performed tests [49]. This approach guarantees a false positive rate of α after all tests. For example, in the example above, with 20 tests and $\alpha = 0.05$, you'd only reject a null hypothesis if the p-value is less than 0.0025. The Bonferroni correction tends to be a bit too *conservative* in rejecting null hypotheses in the interest of controlling the false positives. To demonstrate this, let's calculate the probability of observing at least one significant result when using the correction just described:

$$\begin{aligned}\Pr(\text{at least one significant result}) &= 1 - \Pr(\text{no significant results}) \\ &= 1 - (1 - 0.0025)^{20} \\ &\approx 0.0488\end{aligned}$$

We benefit here from assuming that all tests are independent of each other. In practical applications, that is often not the case. The Bonferroni correction can be somewhat conservative if there are a large number of tests and/or the test statistics are positively correlated. This correction approach controls the probability of false positives only. The correction ordinarily comes at the cost of increasing the probability of producing false negatives, and consequently reducing statistical power.

3.4.2 Holm-Benferroni Method

An alternative, less conservative approach in the category of methods which control the *family-wise error rates* is the *Holm-Benferroni* method [53]. The Holm-Benferroni method is a sequential approach whose goal is to increase the power of

the statistical tests while keeping under control the *Type I* error. Let H_1, ..., H_m be a family of hypotheses, with corresponding p-values of P_1, ..., P_m. The *Holm* algorithm works as follows:

- Sort the p-values in the increasing order. Let the sorted values are denoted by $P_{(1)}$, ..., $P_{(m)}$ and let the associated hypotheses be $H_{(1)}$, ..., $H_{(m)}$.
- Let k be the smallest index such that $P_{(k)} > \frac{\alpha}{m+1-k}$, where α is the significance level.
- Reject all the null hypotheses $H_{(1)}...H_{(k-1)}$ (the hypotheses corresponding to indices 1 to $k-1$) and accept the rest.

Example 3 *Assume that the sorted p-values for five null hypotheses* H_1, H_2, H_3, H_4, H_5 *are equal to* 0.005, 0.011, 0.02, 0.4, *and* 0.13, *correspondingly. For a significance level of* $\alpha = 0.05$, *the Holm procedure will adjust the p-values as follows:*

- *Adjusted p-value for* H_1 *is equal to* $0.005 * 5 = 0.025 < \alpha$*: Reject* H_1.
- *Adjusted p-value for* H_2 *is equal to* $0.011 * 4 = 0.044 < \alpha$*: Reject* H_2.
- *Adjusted p-value for* H_3 *is equal to* $0.02 * 3 = 0.06 > \alpha$*: Accept* H_3*, stop, and accept the remaining hypotheses* H_4 *and* H_5.

The Holm-Bonferroni method is more powerful than the regular Bonferroni, and can always be used as a substitute. In our proposed method, we are using this method for adjusting the significance level in all experiments except the experiments on the Motif discovery datasets.

3.4.3 The False Discovery Rate Method

The classic Benferroni and Holm-Benferroni methods are examples of FWER controlling procedures. These procedures intend to control the probability of making one or more false discoveries (Type I errors) among all the hypotheses (a family of hypotheses) when performing multiple hypotheses tests. As a result, the FWER procedures generally take a more conservative approach in selecting significant observations in favour of reducing false positives (*Type I errors*). In contrast, *False Discovery Rate* (FDR) procedures take a less conservative approach in identifying the important few significant observations from thousands of null hypotheses tested.

The modern widespread use of the FDR is believed to stem from, and be motivated by, the development in technologies that allowed the collection and analysis of a large number of distinct variables in several individuals (*e.g.*, the expression level of each of 10,000 different genes in 100 different persons) [11]. As technology made it possible to generate more datasets with relatively small sample sizes (e.g. few individuals being tested) and large numbers of variables being measured per sample (e.g. thousands of gene expression levels, or motifs), the development of alternative control procedures to classic FWER methods became more crucial. In these cases, too few of the measured variables show statistical significance after classic correction for multiple tests with standard multiple comparison procedures.

The FDR is designed to control the expected proportion of incorrectly rejected null hypotheses ('*false discoveries*') [12]. More formally, let the number of errors

when testing m null hypotheses are represented by the random variables summarized in table 3.1, where,

Table 3.1: Random Variables in testing m null hypotheses

	Null is True (H_0)	Alternative is True (H_1)	**Total**
Declared significant	V	S	R
Declared non-significant	U	T	$m - R$
Total	m_0	$m - m_0$	m

- m is the total number of hypotheses tested
- m_0 is the total number of null hypotheses that are true
- $m - m_0$ is the total number of alternative hypotheses that are true
- V is the number of false positive (Type I error, also called "false discoveries")
- S is the number of true positive
- T is the number of false negatives (Type II error)
- U is the number of true negatives
- R is the number of rejected null hypotheses

Based on these notations, the *FDR* is given by

$$FDR = E(\frac{V}{V+S}) = E(\frac{V}{R}) \tag{3.21}$$

The goal of an FDR control procedure is to guarantee that the expected value of false discoveries is less than a fixed rate α.

Most control procedures for *FDR* analyze the sequence of null hypothesis tests H_1, H_2, ..., H_m, and their corresponding p-values P_1, P_2, ..., P_m. Then, a condition is required to call a null hypothesis statistically significant whenever its corresponding p-value is less than or equal to some threshold, $t \in (0, 1]$. This threshold can be fixed or data-dependent, and the procedure for determining the threshold involves quantifying a desired error rate.

Let us denote the sorted p-values by the sequence $P_{(1)}$, $P_{(2)}$, ..., $P_{(m)}$. There are two main approaches to controlling the FDR. The first one, due to Benjamini and Hochberg [12], seeks to find a cutoff t given the target FDR proportion α. The second one, due to John Storey [101], estimates the FDR given a cutoff value.

The Benjamini–Hochberg Procedure

In one of the earliest control procedures, called the *Benjamini-Hochberg* procedure [12], the p-values are sorted in increasing order and an index k is estimated such that by rejecting all null hypotheses corresponding to p-values $P_{(1)}$, $P_{(2)}$, ..., $P_{(k)}$ the FDR will be less than α. More specifically, for a target false discovery rate α and the total number of m tests, the procedure works as follows:

1. Find the largest k such that $P_{(k)} \leq \frac{k}{m}\alpha$

2. If such a k exists, reject all $H_{(i)}$ for $i = 1, ..., k$. Otherwise, reject nothing.

Example 4 *For a false discovery rate $\alpha = 0.05$, the results of the Benjamini-Hochberg procedure is shown in the table 3.2:*

Table 3.2: Example run of the *Benjamini-Hochberg* procedure

Rank (j)	P-Value	$\frac{j}{m} \times \alpha$	Reject H_0?
1	0.0008	0.005	Y
2	0.009	0.010	Y
3	0.165	0.015	N
4	0.205	0.020	N
5	0.396	0.025	N
6	0.450	0.030	N
7	0.641	0.035	N
8	0.781	0.040	N
9	0.900	0.045	N
10	0.993	0.050	N

The Storey Procedure

A follow-up approach was proposed by Storey *et al.* [101] to take into account the distributions of all p-values and increase the discovery power of the Benjamini-Hochberg procedure. The *Storey* method is an example of a technique in which the point FDRs are calculated for a given cutoff value. In this method, Stoery *et al.* introduce the notion of a *q-value* as the FDR analog to a p-value. A *q-value* is calculated corresponding to each p-value, and each *q-value* Q_i gives us the maximum FDR that we should expect if we reject all the null hypotheses corresponding to smaller q-values than Q_i. In other words, the *q-value* for a given $P_{(i)}$ is defined to be the minimum FDR at which the test is called significant:

$$q\text{-}value(P_{(i)}) = \min_{t > P_{(i)}} FDR(t) \tag{3.22}$$

The reason for using the minimum value in the above formula is that unlike the *type I* error, the FDR is not necessarily strictly increasing with an increasing significant threshold. Hence, the minimum is used to accommodate this property.

For example, in testing a microarray for differential expression of genes, if gene X has a *q-value* of 0.013, it means that 1.3% of genes that show p-values at least as small as gene X are expected to be false positives.

3.5 General Problem Statement

Intuitively, our goal is to find subsequences in a symbolic data sequence S that occur more frequently than expected — based on some model of how sequences like S would look like under "normal conditions", *i.e.*, without containing the "deviating" subsequences we are looking for.

To formalize this notion, we can conceptually model the sequence S, consisting of symbols of a finite alphabet set Σ, as being generated by some random process Θ, in which, at certain positions, subsequences may occur that are generated by a process different from Θ. We can think of Θ as describing the normal behaviour of the process, and we refer to Θ in the rest of the paper as the "*background*" distribution. We can think of the set of the subsequences of S that are not generated by Θ as anomalies in the process modelled by Θ (*e.g.*, failures or intrusions), and we refer to these subsequences, which we want to identify, as *deviating patterns*.

It is necessary to distinguish between a subsequence u as a pattern and its occurrences (or instances) in a sequence S. For instance, given the alphabet $\{1, 2, ..., 9\}$, $u =$"123" is a pattern occurring in $S =$"123764123913" at positions 0 and 6.

We do not make any specific assumptions about the length of the deviating patterns nor the characteristics of the process that generates them. The only assumption is that the anomalies are generated by processes, which are sufficiently different from the background distribution. Under this condition, *i.e.*, in the presence of anomalies, it is expected that the characteristics of a sequence produced by the background distribution will, in general, have changed in a way that allows us to detect those anomalies as deviating patterns. However, we can also expect that, in addition to the embedded deviating patterns, many other subsequences of S (of different lengths), which are partly generated by the background distribution but partly overlap with deviating patterns, may also look deviating. Such patterns are a form of *redundant* deviating patterns. In fact, this phenomenon poses a challenge when trying to identify the truly embedded, deviating patterns. We will demonstrate this challenge after we introduce a more precise (but in parts still very general) formulation of our problem and describe our general approach.

General Problem Statement: Given a symbolic data sequence S and a model Θ from which sequences can be generated, find a set of patterns O with the following properties:

1. the patterns in O have instances in S;
2. the patterns in O are (with high probability) not generated by Θ;
3. every part of S that does not belong to the instances of O is generated with high probability by Θ;
4. among all sets that have properties 1 to 3, O is a smallest set (in terms of cardinality) of patterns with these properties.

In general, we do not know the background distribution for a sequence S, nor do we know the distributions that possibly generate deviating patterns in S. We assume instead that we have access to some "reference" or "training" sequences *like* S, which can be considered to represent "normal behaviour" of the process, and from which we can learn a model $\hat{\Theta}$.

To represent an unknown background distribution for a sequence, we use a *Markov chain* model of some order m. Markov chain models have been widely used in different fields to capture the distribution of data within sequences. A Markov chain of order m models the dependency between symbols by the conditional probability of seeing a symbol at a certain position i in a sequence, given the subsequence

of m symbols occurring at positions $i-1, \ldots i-m$. We denote a Markov chain model of order m by Θ_m.

The approach we propose to detect deviating patterns is based on statistical hypothesis testing. Our *Null Hypothesis* H_0 is that a given data sequence S is generated only by a background distribution, described by a stationary Markov chain model Θ_m. We can then estimate a *p-value* for a given subsequence u that occurs in S t times, which is the *probability* that u occurs at least t times in S under the Null Hypothesis. The p-value is then compared with a *significance level* α, a typically low threshold (*e.g.* 0.01 or even lower); if the p-value is less than α, which means that the probability that the occurrences of u have been generated by Θ_m is very low, the null hypothesis is rejected and the results is said to be *statistically significant*. Throughout this paper, a subsequence that has a lower p-value than a given significance level threshold is simply refereed to as a *significant pattern*.

In this approach, the mentioned problem of redundant deviating patterns becomes the problem of redundant significant patterns. Many patterns may pass the significance test only because they overlap with one of the truly embedded, deviating patterns, and determining which of all the significant patterns constitute the true deviating patterns is a challenge.

Example 5 *To illustrate the problem, consider a time series sequence of length 10,000 generated by a random walk model given by the formula* $y(t) = y(t-1) + \lambda$, *where* $y(1) = 0$, *and* λ *is drawn from a random distribution with mean* $\mu = 0$ *and standard deviation* $\sigma = 2$. *Suppose the time series data is discretized using SAX [69] with input parameters segment size and alphabet size set to* 8 *and* 6, *respectively. The result is a sequence of size* 1250 *with symbols in the set* $\{1, 2, ..., 6\}$. *Suppose the Markov model parameters (i.e. transitional probablilities and stationary probabilities) are learned from the discretized data. Now suppose two subsequences of lengths 4 and 6 are inserted (implanted) at random locations in the sequence. Let "6545", and "112233" be the implanted subsequences that are inserted at different locations into S with frequencies 5 and 4, respectively. Consider extracting all subsequences in S of lengths* 2 *to* 10 *and computing the p-value for each subsequence (details of the p-value calculation are described in Subsection 4.1). Figure 3.1 shows the number of statistically significant patterns of lengths 2 to 10, at a significance level* $\alpha = 0.001$. *We can observe that (1) the number of distinct, statistically significant patterns increases when the length of subsequences increases, (2) the bins corresponding to lengths of the implanted patterns (4 and 6) do not stand out in any way from the overall trend (and represent in fact a comparatively low number of patterns), and (3) the number of significant patterns overall is considerably larger than 2, the number of implanted patterns. Table 3.3 shows the implanted pattern "112233" of length 6 and two other patterns "1122331" and "1223311" of lengths 7 as well as their frequency and p-value, all of which are very similar.*

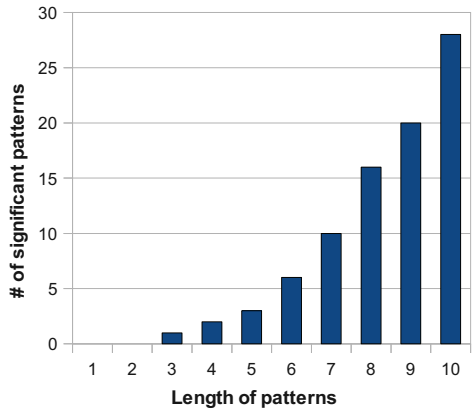


Figure 3.1: *Increasing number of significant patterns with length.*

Table 3.3: Significant patterns of different lengths, with their p-values and frequencies

Length	Pattern info (subsequence, frequency, pvalue)
6	(**'112233'**, 3, 4.16722E-9)
7	(**'1122331'**, 3, 3.11404E-9)
7	(**'1223311'**, 3, 3.11404E-9)

Chapter 4

Proposed Method

As discussed in Section 3.5, the traditional method of testing the significance of all candidate patterns will result in a solution with significant number of redundant patterns, from which the 'true' anomalous patterns cannot be revealed. To address this problem, we propose a method for capturing the relationship between embedded significant patterns in a sequence and those patterns in the sequence that are statistically significant only because they overlap with embedded patterns.

Our assumption is that a sequence S, consisting of symbols of a finite alphabet set Σ, is generated by a normal process Θ, in which, at certain positions, subsequences may occur that are generated by a process different from Θ. The background distribution of the sequence S is modelled by a Markov chain of order m (*e.g.* Θ_m).

If we would know the set of embedded patterns E, we could try to "explain" the statistical significance of a pattern u in S by this set E. The main intuition behind this idea of an explain relationship is to devise a statistical test for the frequency of u with respect to a different Null Hypothesis; the new Null Hypothesis assumes that a set of sequences are generated by Θ_m, with the additional constraint that each sequence must contain the patterns in a set E at the exact same locations as they occur in S.

Definition 6 *Given a set of patterns E and the instances of those patterns in a sequence S, let the total number of the instances be K, and suppose each instance has a start index i_j and a length l_j, $1 \leq j \leq K$. The constraint set $C_{E,S}$ on an arbitrary sequence $S^{'}$ of length $|S|$ is the conjunction of constraints as follows $\bigwedge_{1\leq j\leq K}(\bigwedge_{0\leq x\leq l_j}(S^{'}[i_j+x] = S[i_j+x]))$ (**i.e.** the sequence $S^{'}$ has the same symbols as S at locations of instances in E $\big(S^{'}[i_1] = S[i_1] \wedge S^{'}[i_1+1] = S[i_1+1] \wedge ... \wedge S^{'}[i_1+l_1] = S[i_1+l_1]\big) \wedge \big(S^{'}[i_2] = S[i_2] \wedge ... \wedge S^{'}[i_2+l_2] = S[i_2+l_2]\big) \wedge ... \wedge \big(S^{'}[i_K] = S[i_K] \wedge ... \wedge S^{'}[i_k+l_k] = S[i_k+l_k]\big)$).*

To illustrate this concept, assume an alphabet set $\{1, \ldots, 9\}$, a model Θ_m, and a data sequence S ="12135443512132351" of length 17. When computing the p-value of a subsequence u which is observed t times in S (*e.g.*, u="12", $t = 2$) with respect to Θ_m we consider the probability of having at least 2 occurrences[1] of u in *all* sequences that can be generated by Θ_m and that have the

[1]Throughout the text, the *frequency* and the *number of occurrences* of a pattern are used interchangeably and refer to the number of times a pattern is observed in a sequence

form $S' =_{\_,\_,\_,\_,\_,\_,\_,\_,\_,\_,\_,\_,\_,\_,\_,\_,\_}$ (*i.e.*, the only constraint is the length of the sequence). In contrast, given a set of patterns $E = \{121, 44\}$ with their instances occurring at positions $\{0, 5, 9\}$, and with the constraint set $C_{E,S}$, we consider all sequences of length 17 of the form $S'' =121\_,\_,44\_,\_,121\_,\_,\_,\_,\_$, where the symbols at "open" positions are generated by Θ_m. The constraint set $C_{E,S}$ is specified by $(S''[0] = S[0]) \wedge (S''[1] = S[1]) \wedge (S''[2] = S[2]) \wedge (S''[5] = S[5]) \wedge (S''[6] = S[6]) \wedge (S''[9] = S[9]) \wedge (S''[10] = S[10]) \wedge (S''[11] = S[11])$.

The p-value of a pattern u with frequency t under a Null Hypothesis *with constraint set* E, is the probability $P = Prob(frequency(u) \geq t|C_{E,S})$ that the pattern u occurs at least t times in a sequence that satisfies the constraint set $C_{E,S}$, and is otherwise generated by Θ_m. Given this probability, we define an explain relationship $\succ$ as follows:

Definition 7 *Given a model* Θ_m *and a sequence* S*, we say a* set E explains *(the significance of) a pattern* u *with frequency* t *in* S*, formally* $E \succ u$*, iff* $Prob(frequency(u) \geq t|C_{E,S})$ *is higher than the given significance level* α*. When* E *explains all patterns* u *in a set of patterns* U*, i.e., if* $\forall u \in U : E \succ u$*, we also say that* E *explains* U *and write* $E \succ U$*.*

Example 8 *Assume that the patterns "535", "44334334", and "555355" are found to be statistically significant with the significance level of* $\alpha = 0.01$ *in a synthetic dataset generated with a similar settings to those of Example 5. The p-value of pattern "555355" with the number of occurrences of* 4 *without a constraint is equal to 2.104E-6, which is less than* α*. Considering a constraint set* $C_{E,S}$*, where* $E =$ *{"535", "44334334"}, and calculating the p-value in the presence of the constraint set* $C_{E,S}$ *is equal to* 0.6218*. More formally,*

$$Prob(frequency(\text{“}555355\text{”}) \geq 4|C_{E,S}) = 0.6218 \tag{4.1}$$

The fact that the computed probability is greater than α *implies that* $E \succ \{\text{“}555355\text{”}\}$*.*

We define the *cover set* for a given subset A of patterns as follows:

Definition 9 *Given a model* Θ_m*, a sequence* S *and the set* O *of significant patterns of* S*, the* covering set *of a subset* $A \subseteq O$ *is defined to be the set of all patterns in* O *which can be explained by* A*. This is denoted by* $\mathcal{C}_A$*. More formally,* $\mathcal{C}_A = \{o \in O|A \succ \{o\}\}$*.*

This explain relationship allows us to define a non-redundant set of significant patterns of possibly variable-lengths as a subset of significant patterns in S that explains the statistical significance of all significant patterns in S. It is easy to verify that $E \succ E$, and it is clear that there may be several sets of patterns E that can then explain all significant patterns. For our purpose, we are interested in explaining sets E that are solutions to our general problem statement and satisfy the minimality condition.

Definition 10 *Given a model* Θ_m *and a sequence* S*, a* core pattern set *is a set* E *that explains all significant patterns in* S*, and that has the minimum cardinality among all such sets.*

Finding a core pattern set can be formulated as an optimization problem in which we look for the smallest set of significant patterns that explains all other significant patterns. To solve this problem efficiently we propose a greedy algorithm for constructing an approximate solution incrementally. Given our specific approach to testing the significance of subsequences, and to determining "explain" relationships between them, we can formulate a more specific version of our problem statement:

Specific Problem Statement: Given a symbolic data sequence S and a Markov model Θ_m of order m (which could have been learned from reference sequences), find a set of subsequence patterns O with the following properties:

1. the patterns in O have instances in S;
2. the patterns in O are statistically significant assuming Θ_m as the generating model;
3. every statistically significant subsequence of S with respect to Θ_m is either in O, or is not statistically significant with respect to (Θ_m, C_O), i.e., when given the occurrences of patterns from O in S as additional constraints on the sequences generated by Θ_m;
4. O is a set of smallest size that have properties 1 to 4.

The overall method that we propose to solve this problem, after we estimate the parameters of a Markov model of order m (*i.e.* stationary and transition probabilities) from a '*training*' sequence, consists of the following steps:

1. Extract all subsequences in a desired range of lengths from the data sequence S, using a sliding window, and determine their frequencies in S.
2. For every extracted subsequence u with frequency t, calculate the p-value $Prob(frequency(u) \geq t)$, *i.e.*, the probability that the pattern u occurs in a sequence of length $length(S)$, generated by model Θ_m, at least t times.
3. Add all subsequences whose p-value is smaller than the significance level α (after adjustment for multiple statistical tests) to the set of significant patterns P_{sig}.
4. Find a core pattern set $E \subseteq P_{sig}$.

The steps of our proposed framework for finding surprisingly frequent patterns in sequence data are shown in Figure 4.1. The details of computing p-values and finding a core pattern set are in the next two sections.

4.1 Computing P-Values

The problem of computing p-values of a pattern in a sequence, assuming a Markov model as the background distribution, has been studied in previous work (*e.g.* [25, 83,125]). To the best of our knowledge, this problem has not yet been investigated for models where the generated sequences are additionally constrained by a constraint set. Here we propose to extend the method by Robin *et al.* [90] to derive a formula

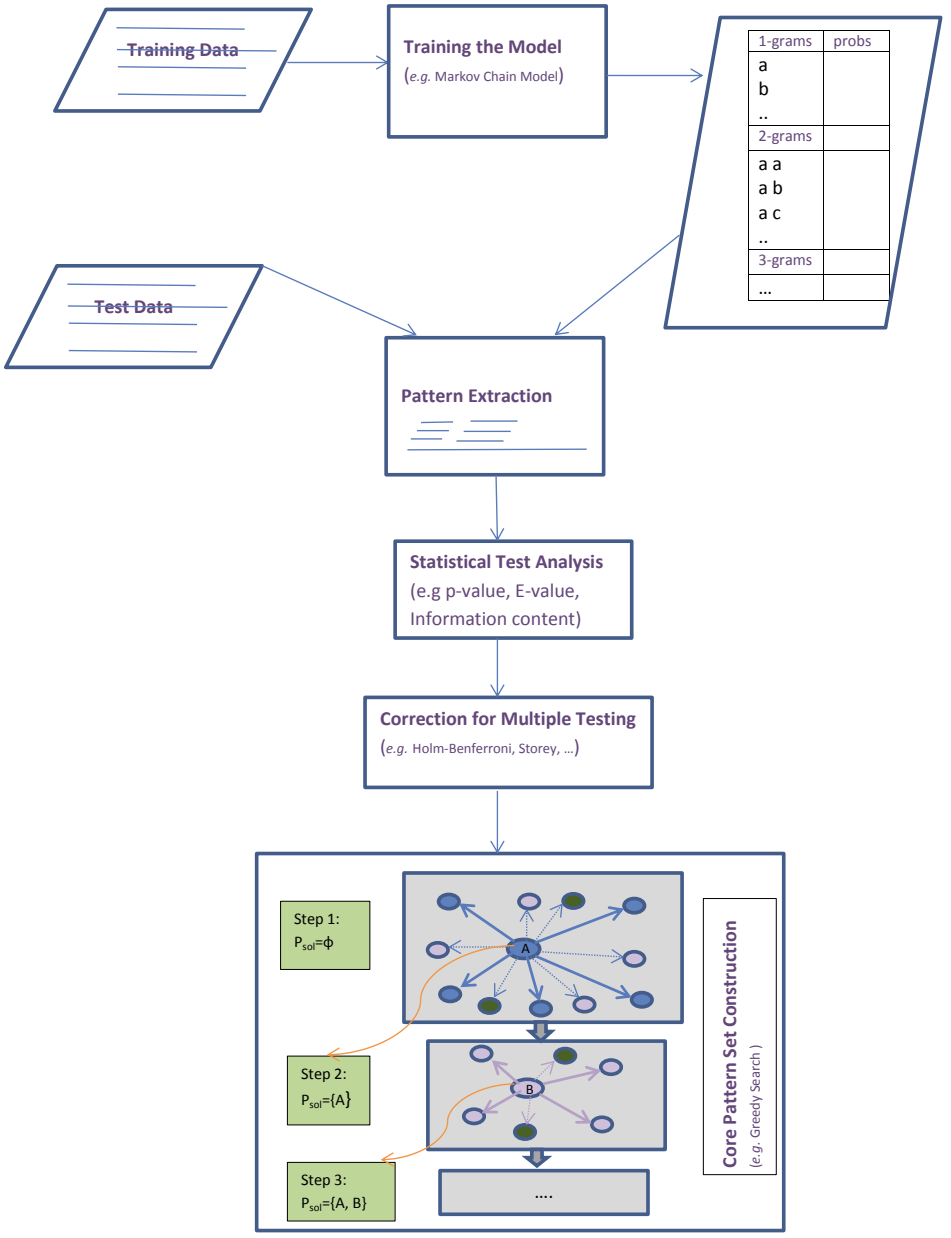


Figure 4.1: *Proposed framework for finding surprisingly frequnet patterns in sequence data.*

for p-values with constrained sequences. Let the function $f_W(j,n)$ be the probability (w.r.t. Θ) that the n^{th} occurrence of a pattern W of length k occurs at index j in a data sequence S. Assume that Θ is a Markov model of order m, and assume for the moment that we already know how to calculate the function f. Then, the p-value of an arbitrary pattern W with observed frequency t (w.r.t. Θ) can be computed as follows:

$$p\text{-}value(W,t) = \sum_{j=1}^{l-k+1} f_W(j,t) \tag{4.2}$$

where l represents the length of the data sequence S, and k represents the length of the pattern W. Intuitively, equation 4.2 gives the sum of probabilities that the t-th occurrence of W is at any location in S (*i.e.*, the probability that the t-th occurrence of W is at location 1, or 2, ..., or at location $l-k+1$). These are the probabilities of mutually exclusive events, and therefore the sum in equation 4.2 is equal to the probability of observing at least t occurrences of the word W.

In order to use this scheme for computing p-values given a constraint set $C_{E,S}$, we have to compute the probabilities of patterns at different locations given the instances of the patterns in E —which are assumed to occur in any generated sequence at the same locations they occur at in S. For simplicity, we derive the formula for a Markov chain of order 1.

Let W_j^n represent the event that the n^{th} occurrence of a pattern W occurs at index j and assume that W_j represents the event that the pattern W occurs at location j, when generated under the constraint set $C_{E,S}$ and the model Θ. The event W_j under these conditions can be decomposed into the following (mutually-exclusive) events:

1. The occurrence of W at location j is its n^{th} occurrence.
2. The occurrence of W at location j is its m^{th} occurrence, for $m < n$.
3. The occurrence of W at location j is its m^{th} occurrence, for $m > n$. This is the event that the n^{th} occurrence of W is at some location $i < j$ and there is another occurrence of W at location j.

Thus, the following equation holds for all $n \in \{1, ..., |S|\}$:

$$P(W_j|C_{E,S}) = P(W_j^n|C_{E,S}) + \sum_{m=1}^{n-1} P(W_j^m|C_{E,S}) + \sum_{i=1}^{j-1} P(W_i^n \cap W_j|C_{E,S}) \tag{4.3}$$

where the terms in the right-hand side of the equation, in the specified order, correspond to the items (1), (2), and (3) of the decomposition, respectively. The last term in Eq. 4.3 can be rewritten as follows:

$$\sum_{i=1}^{j-1} P(W_i^n \cap W_j|C_{E,S}) = \sum_{i=1}^{j-1} (P(W_i^n|C_{E,S}) \times P(W_j|W_i, C_{E,S}))$$

Let us denote the probability $P(W_j^n|C_{E,S})$ by the function $f'_W(j,n|C_{E,S})$; then, based on the equation 4.3 we can use the following recurrence formula for computing $f'_W(j,n|C_{E,S})$:

$$
\begin{aligned}
f_W^{'}(j,n|C_{E,S}) = \quad & P(W_j|C_{E,S}) - \textstyle\sum_{m=1}^{n-1} f_W^{'}(j,m|C_{E,S}) - \\
& \textstyle\sum_{i=1}^{j-1} f_W^{'}(i,n|C_{E,S}) \times P(W_i|W_j,C_{E,S}) \\
f_W^{'}(1,1|C_{E,S}) = \quad & P(W_1|C_{E,S})
\end{aligned}
$$

This recurrence formula allows us to compute the function $f_W^{'}(j,n|C_{E,S})$ using dynamic programming if we know how to compute $P(W_j|C_{E,S})$, which we will explain in the following.

Let $I = \{0,1,...,|S|-1\}$ represent the set of all indices in S, and let I_E represent the set of indices occupied by instances of patterns in E. This means that all indices in I_E are assumed to have known symbols, and the unknown symbols in locations $I - I_E$ are generated by the distribution Θ_m. Let S' be an arbitrary sequence of length $|S|$ that satisfies the constraint set $C_{E,S}$ and is otherwise generated by the distribution Θ_m. Assuming a Markov chain of order 1, the probability of occurrence of an arbitrary symbol c at location i of S' (*i.e.* $S'[i] = c$) , given $C_{E,S}$, can be specified as follows:

1. $i \in I - I_E$: In this case, the probability just depends on the symbol located at the largest index of I_E before i (*i.e.* the last symbol of the closest instance of a pattern before i) and the symbol located at the smallest index of I_E after i (*i.e.* the first symbol of the closest instance of a pattern after i).

2. $i \in I_E$: In this case the probability equals 1, if c matches with symbol $S[i]$; and 0, otherwise.

Stated more formally, for a specific location i, let l_i be the largest index in I_E for which $l_i \leq i$ and u_i be the smallest location index in I_E for which $u_i \geq i$. Then, the probability (w.r.t. $(\Theta_1, C_{E,S})$) of seeing an arbitrary character c at location i in the sequence S' can be formalized as follows:

$$
P(S^{'}[i] = c\,|C_{E,S}) = \begin{cases} P(\mathrm{S}^{'}[\mathrm{i}]{=}\mathrm{c}|(S'[l_i]{=}S[l_i], S'[u_i]{=}S[u_i])) & \textbf{if } l_i < i < u_i \\ 1 & \textbf{if } ((i \in I_E) \wedge (c = S[i])) \\ 0 & \textbf{if } ((i \in l_E) \wedge (c \neq S[i])) \end{cases} \tag{4.4}
$$

For a Markov chain of order 1, this property allows a more efficient implementation by limiting the dependency of occurrences of a character c to at most two known symbols occurring before and after c. The extension to higher-order Markov chains is straightforward. Having a formula for computing $f_w^{'}(x,n|C_{E,S})$ (using dynamic programming), the p-value with respect to $(\Theta, C_{E,S})$ of any pattern W with frequency t can be calculated as:

$$
p\text{-}value(W,t|C_{E,S}) = \sum_{j=1}^{l-k+1} f_w^{'}(j,t|C_{E,S}) \tag{4.5}
$$

4.2 Computing a Core Pattern Set

A *Core Pattern Set* should be computed from the set of significant patterns, after correction for multiple testing, which are the statistical tests performed for evaluating the significance of different subsequences. A naive, exhaustive search for a core pattern set is computationally expensive (exponential in the number of significant patterns) and impractical even for a dozens of patterns. We present an algorithm for constructing an *approximate* solution for a set of significant patterns P, following a greedy forward selection strategy. The approximate solution P_{sol} is constructed by adding one pattern at a time from P. At each step, let P_{rest} be the set of patterns that cannot be explained by the current solution P_{sol}. We choose a pattern from P_{rest} that explains the largest number of remaining patterns. In other words, we select a pattern q^* so that $\{q^*\} \cup P_{sol}$ explains at least as many of the remaining patterns P_{rest} than any other choice. In case of a tie, a pattern whose length is not longer than any other candidate is selected (If several patterns satisfy this condition, one pattern is selected randomly) The process continues by removing the newly selected pattern q from P_{rest} and adding it to P_{sol}. Initially, $P_{sol} = \emptyset$ and $P_{rest} = P$; The algorithm terminates when P_{rest} becomes empty. [2]

The following Pseudocode shows the outline of the algorithm.

Greedy approximation of core pattern set
Input: A set P of statistically significant patterns
Output: Approximate core pattern set P_{sol}

$P_{sol} = \emptyset$;
$P_{rest} = P$;
while $P_{rest} \neq \emptyset$ **do**
  $q^* = \emptyset$;
  $nCover^* = 0$;
  //Choosing q^*, the "best" pattern from P_{rest}
  for all $p_i \in P_{rest}$ **do**
    $CS_i = \{p_j \in P_{rest} | (P_{sol} \cup \{p_i\}) \succ p_j\}$
    if $(|CS_i| > nCover^*)$ OR $(|CS_i| = nCover^*$ AND length(p_i) <length$(q^*))$
    then
      $q^* = p_i$;
      $nCover^* = |CS_i|$;
    end if
  end for
  $P_{sol} = P_{sol} + \{q^*\}$;
  $P_{rest} = P_{rest} - \{q^*\}$;
end while
return P_{sol};

---

[2] We have not investigated the approximation factor of the proposed greedy algorithm theoretically. This is left as an open problem for future work.

4.3 Smoothing Model Parameters

Computing the probability of observing a symbol given a history of symbols before it (*i.e.* the conditional probability $p(w_i|w_{i-l+1}...w_{i-1})$) is a required calculation in testing the *explain relation* between statistically significant patterns. More specifically, for two patterns $u =$"*abc*" and $v =$ "*abcd*" which happen to be statistically significant after correction for multiple testing, we need to estimate the probability $p(v|u)$ in order to evaluate the *explain* relation $u \succ v$. On the other hand, estimating the conditional probabilities based on the *Maximum likelihood* (*ML*) model highly depends on the training data. Assume that the substring "abcd" is not seen in the training data (*e.g.* all the occurrences of the string "abc" are followed by symbols other than 'd'), but it is likely to observed in the test data. According to a pure maximum-likelihood estimator these events would have probability zero, which is plainly wrong since previously unseen events are likely do occur in independent test data, and the maximum-likelihood model does not provide a correct estimation This means that some probabilities might be evaluated to 0 just because a specific sequence of symbols are not seen in the training data. This is particularly very likely if we are dealing with some 'surprising' patterns, and their overlapping patterns. Conditional probabilities with zero values make *explain* relation evaluation problematic because it makes a potential 'true surprising pattern' incapable of *explaining* a containing pattern. In the example of two patterns u and v mentioned above, and for a Markov chain of order 2, the probability of observing an instance of pattern v at index i in sequence data, given that an instance of pattern u is already observed at index i, is calculated as follows:

$$p(v|u) = \frac{frequncy(abcd)}{frequency(abc)} \tag{4.6}$$

The probability $p(v|u)$ will be equal to 0 if the substring "abcd" is not observed in the training data, and as a result, the explain relation test for $u \succ v$ cannot be performed properly. Note that in practice, the pattern u is likely to explain the frequency of the pattern v.

A *smoothing* technique is required to address this problem by adjusting the probabilities. As discussed in Section 3.2, the main idea of smoothing is to adjust low probabilities such as zero probabilities upward, and high probabilities downward. In our proposed model, we used the backoff version of the Witten-Bell smoothing presented in Section 3.2.4 to adjust the probabilities due to its efficiency in a recursive implementation using dynamic programming.

4.4 Example Run of the Core Pattern Set Algorithm

In this Section, we will demonstrate an example run of our Core Pattern Set algorithm on a synthetic data set with implanted motifs, with a similar approach to what described in Chapter 3

Example 11 *A time series sequence of length* 10,000 *is generated by a random walk model given by the following formula*

$$y(t) = y(t-1) + \lambda \tag{4.7}$$

where $y(1) = 0$*, and* λ *is drawn from a random distribution with mean* $\mu = 0$ *and standard deviation* $\sigma = 2$*. The time series data is discretized using SAX [69] with input parameters segment size and alphabet size set to* 10 *and* 5*, respectively. The result is a sequence of size* 1000 *with symbols in the set* $\{1, 2, ..., 5\}$*. The resulting dataset is used as the training dataset for learning the parameters of the Markov chain model and will be referred to as* S_{train}*. The test dataset* S_{test} *is generated by inserting (implanting) two subsequences of lengths* 3 *and* 6 *at random locations in the original sequence* S_{train}*. Let "535" and "132221" be the implanted subsequences that are inserted at (different) random locations into* S_{train} *with frequencies* 4 *and* 3*, respectively.*

4.4.1 Step 1: Learning the Parameters of the Markov Chain Model

In the first step of our Core Pattern Set algorithm, we use the training data to learn the parameters of a Markov Chain model of order 1. We use the backoff version of the Witten-Bell smoothing technique presented in Section 3.2.4 to smooth the 1-gram and bigram probabilities. The probability values for 1-gram and bigrams and the backoff values for 1-grams generated by the Wittern-Bell technique are shown in the table 4.1.

4.4.2 Step 2: Extracting Subsequences

In the second step of the algorithm, we extract all the unique subsequences in the length range between 3 and 15 in the test sequence (S_{test}) and compute the observed frequencies of these subsequences. This step is straightforward.

4.4.3 Step 3: Computing P-values and Identifying the Significant Patterns

In the third step of the algorithm, we compute the p-value for each subsequence using the techniques presented in Section 4.1. Using a significance level $\alpha = 0.01$, a total number of $m = 95$ patterns are found to have a p-value less than α. Following that, we apply the *Holm-Benforreni* method to correct for multiple hypothesis testing. The first few corrections using the *Holm-Benforreni* method are shown in the table 4.2.

After correction for multiple hypotheses testing, only 45 (out of 95) patterns are selected by our algorithm for which the null hypothesis is rejected.

4.4.4 Step 4: Greedy Search Algorithm for Finding a Core Pattern Set

In the next step of the algorithm, we use a greedy search strategy, as outlined in the Pseudocode 4.2, to find a Core Pattern Set from the set of significant patterns. In the beginning of the greedy search algorithm, P_{sol} is empty (*i.e.* $P_{sol} = \emptyset$), and the set P_{rest} consists of the all the 45 significant patterns.

After the pair-wise *explain relation* testing between patterns, the pattern which explains the highest number of remaining patterns is selected as the *best* pattern in this step of the search (q^*). In our running example, the pattern "44334334″ is selected as the *best* pattern by explaining 17 pattern among the remaining significant

Table 4.1: Probabilities and backoff values generated by the Witten-Bell Smoothing technique in the log scale

Probability (log)	n-gram	Backoff (log)
1-grams:		
-0.7675011	1	-1.664769
-0.530273	2	-1.551823
-0.7526095	3	-1.366187
-0.8507411	4	-1.351233
-0.6685757	5	-1.66921
-2.702	<BOS>	
-99	<EOS>	-0.196177
2-grams:		
-0.03938902	1 1	
-1.124103	1 2	
-1.361728	2 1	
-0.05571544	2 2	
-1.174641	2 3	
-0.9542425	3 2	
-0.1029842	3 3	
-1.079181	3 4	
-0.9822713	4 3	
-0.08648048	4 4	
-1.255273	4 5	
-1.435367	5 4	
-0.02458927	5 5	

patterns and including itself. The conditional p-values for some of the significant patterns with the constraint set $C_{E,S_{test}}$, where $E = \{\text{“}44334334''\}$ are shown in the table 4.3. In table 4.3, the first column represents the pattern p_i for which the *explain* relation $E \succ p_i$ is investigated, the second column represents the conditional p-value in the presence of the constraint set $C_{E,S_{test}}$, and the last column represents the result of the *explain* relation, taking the values of 'Y' (*explains*) or 'N' (*does not explain*).

At the end of the step 1 of the greedy search algorithm, pattern "44334334" is selected as the *best* pattern (q^*) and will be added to the set P_{sol}, and the 17 patterns which are explained by this pattern are removed from the P_{rest}, leaving only 28 ($= 45 - 17$) patterns in the set P_{rest}.

The second step of the greedy search algorithm starts with $P_{sol} = \{\text{“}44334334''\}$ and updated set P_{rest} which has 28 patterns. In this step, the pattern "535″ is selected as the *best* pattern by explaining 14 patterns among the remaining significant patterns in P_{rest}. The conditional p-values for some of the significant patterns with the constraint set $C_{E,S_{test}}$, where $E = \{\text{“}44334334'', \text{“}535''\}$ are shown in the table 4.4.

At the end of the step 2 of the greedy search algorithm, pattern "535" is selected as the *best* pattern (q^*) and will be added to the set P_{sol}, and the 14 patterns which are explained by this pattern are removed from the P_{rest}, leaving only 14 ($= 28 - 14$)

Table 4.2: Example run of the *Holm-Benforreni* procedure

Rank(j) and Pattern	P-Value	$m+1-j$	Adjusted P-value	Reject H_0?
1: "555535"	0.00000201	95	0.000190	Y
2: "535555"	0.00000201	94	0.000190	Y
3: "555355"	0.00000210	93	0.000195	Y
4: ...	...	92	...	...

patterns in the set P_{rest}.

The third step of the greedy search algorithm starts with $P_{sol} = \{$"44334334″, "535″$\}$ and updated set P_{rest} which has 14 patterns. In this step, the pattern "1322212″ is selected as the *best* pattern by explaining all the 14 remaining patterns. The conditional p-values for the significant patterns with the constraint set $C_{E,S_{test}}$, where $E = \{$"44334334″, "535′, "1322212‴$\}$ are shown in the table 4.5.

At the end of the step 3 of the greedy search algorithm, pattern "1322212" is selected as the *best* pattern (q^*) and will be added to the set P_{sol}. The algorithm stops as the set P_{rest} will be empty at the end of this step, and the set P_{sol} consists of the patterns "44334334", "535", and "1322212", which explain all the significant patterns in the test sequence S_{test}.

The set of significant patterns and the final Core Pattern Set are shown in the table 4.6. The set of all significant patterns consists of 45 patterns while the core pattern set contains only 3 patterns. The set of significant patterns can be divided into three groups: 1) The groups of patterns that has the pattern "132221" as a substring, the group of patterns that contain the pattern "535", and another group that included the pattern "44334334" as a substring. The first two substrings are exactly the patterns implanted in the test data during data generation. Our investigation shows that the pattern "44334334" stands as a significant pattern in the original random walk data sequence (S_{train}) even without implanting any subsequence and is a false positive. It is worth mentioning that the statistically significant patterns are unlikely (less than 0.01 in our experiment) to have occurred solely by chance. However, it is not impossible that some patterns pass the significance tests on the random walk data that, by construction, should not have significant patterns. Excluding the pattern "44334334", the patterns returned by the approximate core pattern set algorithm closely match the implanted motifs.

Table 4.3: Conditional p-value for significant patterns w.r.t $E = \{\text{“44334334”}\}$

Pattern p_i	p-value	$(E \succ \{p_i\})$?	
“444444444334334”	0.0831 ($> \alpha = 0.01$)	Y	
“444444443343344”	0.0832 ($> \alpha = 0.01$)	Y	
“44444444334334”	0.1186 ($> \alpha = 0.01$)	Y	
“44444443343344”	0.1187 ($> \alpha = 0.01$)	Y	
“4444444334334”	0.1693 ($> \alpha = 0.01$)	Y	
“4444443343344”	0.1694 ($> \alpha = 0.01$)	Y	
“444334334”	0.7013 ($> \alpha = 0.01$)	Y	
“443343344”	0.7017 ($> \alpha = 0.01$)	Y	
“43343344”	0.7114 ($> \alpha = 0.01$)	Y	
...	...	...	...
...	...	...	...
“5355555”	1.611E-6 ($< \alpha = 0.01$)	N	
“5555535”	1.611E-6 ($< \alpha = 0.01$)	N	
“5535555”	1.681E-6 ($< \alpha = 0.01$)	N	
“55535”	2.669E-6 ($< \alpha = 0.01$)	N	
“55355”	2.784E-6 ($< \alpha = 0.01$)	N	
“53555”	2.669E-6 ($< \alpha = 0.01$)	N	
“5355”	3.434E-6 ($< \alpha = 0.01$)	N	
“5535”	3.434E-6 ($< \alpha = 0.01$)	N	
“535”	4.414E-6 ($< \alpha = 0.01$)	N	
...	...	...	...
...	...	...	...
“1113222122”	7.2819E-11 ($< \alpha = 0.01$)	N	
“113222122”	9.9217E-11 ($< \alpha = 0.01$)	N	
“111322212”	3.0677E-7 ($< \alpha = 0.01$)	N	
“13222122”	1.3517E-10 ($< \alpha = 0.01$)	N	
“132221222”	3.2844E-7 ($< \alpha = 0.01$)	N	
“11132221”	3.2251E-7 ($< \alpha = 0.01$)	N	
“1132221”	4.3847E-7 ($< \alpha = 0.01$)	N	
“132221”	5.9602E-7 ($< \alpha = 0.01$)	N	

Table 4.4: Conditional p-value for significant patterns w.r.t $E = \{$"44334334″, "535″$\}$

Pattern p_i	p-value	$(E \succ \{p_i\})$?	
"5355555"	0.5356 ($> \alpha = 0.01$)	Y	
"5555535"	0.5322 ($> \alpha = 0.01$)	Y	
"5535555"	0.5334 ($> \alpha = 0.01$)	Y	
"555355"	0.6218 ($> \alpha = 0.01$)	Y	
"55355"	0.7278 ($> \alpha = 0.01$)	Y	
"53555"	0.7309 ($> \alpha = 0.01$)	Y	
"5355"	0.8547 ($> \alpha = 0.01$)	Y	
"5535"	0.8533 ($> \alpha = 0.01$)	Y	
...	...	...	...
...	...	...	...
"1113222122"	4.05E-11 ($< \alpha = 0.01$)	N	
"113222122"	5.542E-11 ($< \alpha = 0.01$)	N	
"111322212"	6.136E-11 ($< \alpha = 0.01$)	N	
"13222122"	7.585E-11 ($< \alpha = 0.01$)	N	
"132221222"	2.224E-7 ($< \alpha = 0.01$)	N	
"11132221"	1.885E-7 ($< \alpha = 0.01$)	N	
"1132221"	2.572E-7 ($< \alpha = 0.01$)	N	
"132221"	3.509E-7 ($< \alpha = 0.01$)	N	

Table 4.5: Conditional p-value for significant patterns w.r.t $E = \{$"44334334″, "535″, "1322212″$\}$

Pattern p_i	p-value	$(E \succ \{p_i\})$?
"1113222122"	0.0392 ($> \alpha = 0.01$)	Y
"113222122"	0.0705 ($> \alpha = 0.01$)	Y
"111322212"	0.0780 ($> \alpha = 0.01$)	Y
"13222122"	0.6218 ($> \alpha = 0.01$)	Y
"11322212"	0.1255 ($> \alpha = 0.01$)	Y
"11132221222"	0.2053 ($> \alpha = 0.01$)	Y
"11113222122"	0.2154 ($> \alpha = 0.01$)	Y
"1132221222"	0.3452 ($> \alpha = 0.01$)	Y
"1111322212"	0.3825 ($> \alpha = 0.01$)	Y
"132221222"	0.7924 ($> \alpha = 0.01$)	Y
"11132221"	0.0825 ($> \alpha = 0.01$)	Y
"1132221"	0.1313 ($> \alpha = 0.01$)	Y
"132221"	1.000 ($> \alpha = 0.01$)	Y

Table 4.6: The set of significant patterns (after correction for multiple testing) and the Core Pattern Set returned by our algorithm

All Significant Patterns
“1113222122”,“113222122”,“111322212”,“13222122”, “11322212”,“1322212”,“11132221222”,“11113222122”, “1132221222”,“1111322212”,“132221222”,“11132221”, “1132221”,“ 132221”, “5355555”,“5555535”,“5535555”,“5555355”,“5553555”, “535555”,“555535”,“553555”,“555355”,“53555”,“55535”, “55355”,“5355”,“5535”,“535”, “444444444334334”,“444444443343344”, “44444444334334”,“44444443343344”, “4444444334334”,“4444443343344”, “444444334334”,“444443343344”,“44444334334, “44443343344”,“4444334334”,“4443343344”, “444334334”,“443343344”, “44334334”,“43343344”
Core Pattern Set
“44334334”,“535”,“1322212”

Chapter 5

Model Extension for Approximate Pattern Matching

In many applications, 'surprising' patterns occur in data sequences with *variations*. Well-known examples are *DNA binding sites* of *transcription factors* (motifs), which vary both in length and sequences of *nucleotides* (building blocks of *nucleic acids*). In this section, we extend our proposed model and framework in order to allow some degree of variation in pattern specification. Thus, our proposed model can be extended to capture a wider class of applications, including some interesting problems in Bioinformatics.

5.1 Extended Motif Model

As it was discussed, the exact nucleotide sequence that is recognized by the same *transcription factor* varies at different binding sites. A *transcription factor* can bind to a number of partially similar looking sequences. Some positions in the binding sequence are highly conserved, which means that base substitutions in these positions can reduce or completely eliminate the binding of the transcription factor. Whereas some other positions in the binding sequence are relatively less conserved and can be mutated without affecting the binding capability. This variation is useful as it allows different degrees of interaction with the transcription factors at different DNA binding sites, which in turn results in different expression levels of various genes regulated by the same transcription factor.

Three types of models have been already used for motif representation:

- **Mismatch String**: is a tuple $< cs, d >$, where cs represents a string from the alphabet $\sum = \{A, C, G, T\}$, and d represents the maximum number of mismatches allowed for a binding site to be considered a *hit*. (match).

  Example 12 *For a mismatch string* $<$ *"*$ACGTGAACG$*"*$, 2 >$*, the strings "TCGTGAACG", and "AAGTGAACT" are hits, however the strings "CATT GAACG" and "ATGTGTTCG" are not considered as hits for this motif.*

- **Position Weight Matrix (PWM)**: The main idea behind the Position Weight Matrix model is that it captures the base preferences at each position

of the binding sequence of the transcription factor. More formally, a PWM is a tuple $< M, t >$, where M is a matrix of $4 \times n$ where n is the length of the motif, and each column j of the matrix represents the probability distribution of the nucleotide vector $< A, C, G, T >$ at position j of the binding sequences. A candidate string of length n is considered to be a *hit* of the multiplications of probabilities in respective rows are greater than the threshold t. Often the elements in PWMs are calculated as *log likelihoods*. That is, the elements of the PWM are transformed using a background probability distribution b so that:

$$M_{k,j} = ln(\frac{M_{k,j}}{b_k}) \tag{5.1}$$

The simplest background model assumes that each letter appears equally frequently in the dataset. That is, $b_k = 1/|\sum|$, where $|\sum|$ is the number of symbols in the alphabet (*e.g.* $\sum = \{A, C, G, T\}$).

Example 13 *Assume that the set of binding sites for a transcription factor are shown in Table 5.1*

Table 5.1: List of binding sites for a transcription factor

GAGGTAAAC
TCCGTAAGT
CAGGTTGGA
ACAGTCAGT
TAGGTCATT
TAGGTACTG
ATGGTAACT
CAGGTATAC
TGTGTGAGT
AAGGTAAGT

The PWM can be derived by computing the frequency matrix and normalizing each column of the matrix, as follows:

$$M = \begin{matrix} A \\ C \\ G \\ T \end{matrix} \begin{bmatrix} 0.3 & 0.6 & 0.1 & 0.0 & 0.0 & 0.6 & 0.7 & 0.2 & 0.1 \\ 0.2 & 0.2 & 0.1 & 0.0 & 0.0 & 0.2 & 0.1 & 0.1 & 0.2 \\ 0.1 & 0.1 & 0.7 & 1.0 & 0.0 & 0.1 & 0.1 & 0.5 & 0.1 \\ 0.4 & 0.1 & 0.1 & 0.0 & 1.0 & 0.1 & 0.1 & 0.2 & 0.6 \end{bmatrix} \tag{5.2}$$

Given the Position Weight Matrix M, the probability of the sequence $S =$ "$GAGGTAAAC$" can be calculated as follows:

$$P(S|M) = 0.1 \times 0.6 \times 0.7 \times 0.1 \times 1.0 \times 0.6 \times 0.7 \times 0.2 \times 0.2 = 0.0007 \tag{5.3}$$

The calculated probability should be compared against the motif threshold to determine if the given string is a hit for the motif or not.

- **Consensus**: is a string ds of length n where each position is a non-empty subset of $\{A, C, G, T\}$. These subsets correspond to the IUPAC (International Union of Pure and Applied Chemistry) symbols for DNA sequences [2] (*e.g.* W stands for $\{A, T\}$). A candidate string s is said to be a *hit* (match) against ds if every position of s is a subset of respective position in ds. Otherwise, it is a *non-hit* (non-match).

  Example 14 *Based on the IUPAC table [2], the motif AYNBR corresponds to* $\{A\}\{C, T\}$ $\{A, C, G, T\}\{C, G, T\}\{A, G\}$.

Among the motif models that have been used for motif representation, we chose to use the *mismatch string* model for representing *generalized surprising patterns*. The reason is that the mismatch string model is more similar to the simple string model used in our basic model, and requires less changes for extension. We represent the model by $Q_{s,d}$, where s represents a string of symbols and d is the maximum number of allowed mismatches, w.r.t the string s. In other words, $Q_{s,d}$ represents the set of all subsequences that deviate from s by at most d mismatches. All instances of the mismatch pattern $Q_{s,d}$ are assumed to be of the same length $|s|$. The definitions presented in Chapter 4 can be adapted to mismatch patterns as well when taking into account that the instances of a *mismatch pattern* $Q_{s,d}$ are allowed to have up to d mismatches w.r.t. string s.

The proposed *mismatch model* poses several challenges, particularly in the way the p-value of a *mismatch pattern* is calculated in the presence of a *constraint set*. We discuss the p-value calculation in two different cases:

1. Calculating the p-value of a *mismatch pattern* in a sequence without a *constraint set*

2. Calculating the p-value in the presence of a *constraint set*

In the first case, we are interested in calculating $Prob(freq(Q_{s,d}) \geq t)$, where t is the observed frequency of the mismatch pattern $Q_{s,d}$. The observed frequency t should be computed considering the maximum number of allowed mismatches d for pattern Q. In Section 4.1, we proposed a method for calculating the exact p-value of a pattern without a mismatch. Calculating the exact p-values for *mismatch patterns* is too time-consuming and impractical in applications with a large number of candidate patterns. The practical alternatives to exact p-value calculation are approximation methods, which are widely used in the literature [92, 98]. In practice, p-values of *mismatch patterns* are estimated using a *Poisson* distribution, a *Normal* distribution, or a Compound-Poisson distribution [16, 91].

5.2 Computing P-values for Approximate Patterns

In this paper, we choose to approximate the p-values of *mismatch patterns* using Poisson distribution because of its fast computation time and fairly accurate results [16, 91]. In this approach, the frequencies of the patterns Q are assumed to be random variables following a Poisson distribution with parameter λ ($N(Q) \sim \rho(\lambda)$). The parameter λ is equal to $n\mu$, where n is the length of the input sequence and

μ is the probability of occurrence of the pattern at any location. For a mismatch pattern $Q_{s,d}$, the probability of the occurrence of the pattern at any location is the sum of probabilities of the occurrences of all the strings $s^{'}$ of length $|s|$ which have at most d mismatches w.r.t. string s. More formally,

$$\mu = Prob(Q_{s,d}|\theta_m) = \sum_{s' \in \Sigma^{|s|}, dis(s',s) \leq d} Prob(s^{'}|\Theta_m) \tag{5.4}$$

where Σ is the alphabet (set of symbols), *dist* is the *edit distance* between two strings, and θ_m is the background distribution of the data sequence. The p-value of the *mismatch pattern* $Q_{s,d}$ with the observed frequency t in a data sequence can be approximated using the following formula [16]:

$$Prob(\rho(\lambda) \geq t) = 1 - e^{-\lambda} \sum_{i=0}^{t-1} \frac{\lambda^i}{i!} \tag{5.5}$$

5.3 Implementing Explain Relations Using Poisson Binomial Distribution

In the presence of a *constraint set*, approximating the p-value of a pattern becomes more complicated because the probability of occurrence of a pattern varies at different positions of a given data sequence. The counts of patterns should be modelled using a distribution which captures trials with different outcomes that can occur with different probabilities. We use the *Poission binomial* distribution to model the count of a mismatch pattern.

The Poisson binomial distribution is the distribution of the sum of independent and non-identical random indicators [54]. Each indicator follows a Bernoulli distribution with individual success probability. When all success probabilities are equal, the Poisson binomial distribution is a binomial distribution. Let I_j, $j = 1, ..., n$, be a series of n random indicators, each following Bernoulli distribution:

$$I_j \sim Benourlli(p_j), \qquad j = 1, ..., n \tag{5.6}$$

where $p_j = Pr(I_j = 1)$ is the success probability for indicator I_j. We will refer to the sequence of probabilities p_j as 'point probabilities'. The Poisson binomial random variable N is defined by $N = \sum_{j=1}^{n} I_j$ and can take any value in $\{0, 1, ..., n\}$. Therefore, the expected value of N is equal to

$$E(N) = \sum_{j=1}^{n} p_j \tag{5.7}$$

For a given *mismatch pattern* $Q_{s,d}$ and a data sequence S, if the probabilities $p_j, 1 \leq j \leq |S|$ are known, the Poisson binomial distribution model of counts provides a means to calculate the *probability mass function* (pmf) $\xi_k = Prob(N = k), k = 1, 2, ..., |S|$ of the pattern count recursively. The recursion is based on two parameters, the count k and the length of the data sequence. We first assume that the *point probabilities* p_j are known and present a method for calculating the probability mass function based on the formula introduced by Hong *et al.* [54].

Let $N_j = \Sigma_{m=1}^{j} I_m$ represent the sum of random indicators I_m up to the index j of the data sequence, where the random indicator I_m is defined as in equation 5.6. Also, let $\xi_{k,j} = Prob(N_j = k)$. In other words, $\xi_{k,j}$ represents the probability that the sum of indicators I_m up to the index j of the data sequence is equal to k. When calculated recursively, the function $\xi_{k,j}$ gives us the desired probability mass function given that $\xi_k = \Sigma_{j=1}^{|S|} \xi_{k,j}$. It can be verified that the function $\xi_{k,j}$ can be defined using the following recursive formula

$$\xi_{k,j} = (1 - p_j)\xi_{k,j-1} + p_j \xi_{k-1,j-1}, \qquad 0 \leq j \leq |S|, 0 \leq k \leq t \tag{5.8}$$

where t is equal to the count for which the probability mass function is desired. The boundary conditions for formula 5.8 are defined as follows:

$$\begin{aligned} &\xi_{-1,j} = \xi_{j+1,j} = 0, \qquad j = 0, 1, ..., |S| - 1 \\ &\xi_{0,0} = 1 \end{aligned} \tag{5.9}$$

Using the probability mass function ξ_k, which can be calculated efficiently using dynamic programming, it is easy to calculate the p-value of a *mismatch pattern* $Q_{s,d}$ with observed frequency t, as follows:

$$p\text{-}value(Q_{s,d}, t|C_{E,S}) = Prob(N(Q_{s,d}) \geq t|C_{E,S}) = 1 - \sum_{k=0}^{t-1} \xi_k \tag{5.10}$$

In the presence of a constraint set $C_{E,S}$, the *point probabilities* should be calculated based on the assumption that some positions of the data sequence S are constrained with instances of patterns specified in the set E. Let I_E represent the union of all indices in the data sequence S that are occupied by any instance of a pattern in E. In order to estimate the probability of occurrence of a *mismatch pattern* $Q_{s,d}$ at an arbitrary index i of the data sequence S, we have to consider the effect of constraints for indices I_E. Let $L_{min}(Q, i|C_{E,S})$ represent the minimum distance between an index i in sequence S and all indices in I_E, either to the left or right of the index i. More formally, $L_{min}(Q, i|C_{E,S})$ is defined as follows:

$$\min\left(\min_{k \in I_E}\{|i - k|\}, \min_{k \in I_E}\{|i + |s| - 1 - k|\}\right) \tag{5.11}$$

in which $i + |s| - 1$ represents the index of the last symbol of a pattern instance $Q_{s,d}$, if it occurs at index i. Also, let $Prob(Q_{s,d}|\theta_m)$ represent the probability of occurrence of a *mismatch pattern* Q in a sequence without any constraint, as specified in equation 5.4. We refer to $Prob(Q_{s,d}|\theta_m)$ as a *constraint-free* probability.

Using the Fundamental theorem of *Markov* chains [81], it can be verified that as the distance $L_{min}(Q, i|C_{E,S})$ is increased, the probability p_i (the point probability of pattern $Q_{s,d}$ at index i) will converge to the *constraint-free* probability $Prob(Q_{s,d}|\theta_m)$. We use this property of a stationary Markov chain model to limit the number of different *point probabilities* that should be calculated for a *mismatch pattern* in a constrained data sequence S. The key idea is that we calculate a specific *point probability* for those indices which are very close to a constraint in the data sequence. In other cases, we approximate the *point probability* of the pattern with its constraint-free probability. More specifically, for a given mismatch pattern $Q_{s,d}$, and for any index i, we calculate $L_{min}(Q, i|C_{E,S})$ and compare it with

a distance threshold d_θ. If $L_{min}(Q, i|C_{E,S}) < d_\theta$, we calculate the *point probability* for index i based on the closest constraint to index i and a similar method that is presented in formula 4.4. Otherwise, we approximate the point probability p_i with the constraint-free probability $Prob(Q_{s,d}|\theta_m)$. In this way, the point probabilities for a *mismatch pattern* are equal to $Prob(Q_{s,d}|\theta_m)$ except at indices which are at distance less that θ_m from a constraint index in I_E.

Chapter 6

Complexity Analysis

6.1 Time Complexity of the Proposed Algorithms

We analyze the complexity of our algorithms separately for the simple string model and the mismatch string model. In the following we make the assumption that we are interested in finding surprising patterns in a test sequence S. The length of the desired patterns are assumed to vary in the range $[\ell_1, \ell_2]$.

6.1.1 Simple String Model

The analysis is done for each step of the algorithm:

- **Statistical Test Analysis**: This step involves computing p-values for all the extracted subsequences from the test sequence. The number of extracted patterns using the sliding windows in a sequence S is of complexity $O((\ell_2 - \ell_1)|S|)$, where $|S|$ denotes the length of the test sequence S. For a pattern q_i of length k_i and the observed frequency t_i, the exact p-value computation is of time complexity $O(k_i t_i |S|)$ [91]. Therefore, the worst-case time complexity of exact p-value computation for extracted patterns is equal to:

$$O(|S| \sum_{i \in \mathcal{W}} k_i t_i) \tag{6.1}$$

  where $\mathcal{W}$ is the set of all the extracted subsequences (patterns) from the sequence S. The number of extracted patterns was shown to be of order $O((\ell_2 - \ell_1)|S|)$. It is reasonable to assume that the length of the patterns are constant with respect to the length of the sequence. Therefore, the specified time complexity can be simplified as $O(|S|^2 \times \sum_{i \in \mathcal{W}} t_i)$.

  In the case that an approximation technique is used for computing the p-values using the *Poisson* distribution, as will be discussed in more details in Section C.2.2, the p-value computation for a pattern q_i with observed frequency t_i can be reduced to $O(t_i)$. Therefore, the worst-case time complexity of p-value computation using the Poisson distribution for all the extracted patterns is equal to:

$$O(|S| \times \sum_{i \in \mathcal{W}} t_i) \tag{6.2}$$

where $\mathcal{W}$ is the set of all the extracted subsequences (patterns) from the test sequence S.

- **Correction for Multiple Testing**: The complexity of this step depends on the method used for correction, but in the case that the correction method involves sorting the computed p-values (*e.g. Holm-Benferroni*, Benjamini-Hochberg procedure), it can be easily verified that the worst-case time complexity of this step is equal to

$$O(C \times log(C)) \tag{6.3}$$

where C represents the number of patterns whose p-value is less than the significance level α.

- **Core Pattern Set Construction**: The time complexity of the core pattern set construction depends on outcomes of the greedy search algorithm in different steps; *e.g.* how many significant patterns are *explained* in each step, and how many of the pattern are passed to the next step. Therefore, we just give the worst-case time complexity by assuming that the number of core patterns returned by the algorithm in the end is equal to R. In the first step of the greedy search algorithm, the explain relation should be tested for all patterns against each other. Assuming that the number of significant patterns (after correction for multiple testing) is equal to N_{sig}, the number of explain relation tests required in the first step of the greedy search is equal to $\frac{N_{sig}(N_{sig}-1)}{2}$. The core part of an explain relation is a p-value computation, which is implemented based on the exact computation method when the underlying model is a simple string. Therefore, the time complexity of the first step of the search algorithm is equal to

$$O(N_{sig}^2 \times |S|^2 \times \sum_{j \in N_{sig}} t_j) \tag{6.4}$$

Based on the assumption that the number of core patterns returned by the algorithm is equal to R, the greedy search is repeated R times even though the number of the input significant patterns to each step of the greedy search is decreased as the search continues (*e.g. the number of patterns in the second step of the search algorithm is less than* N_{sig}). Therefore, the worst-case time complexity of the core patter set construction can be specified as follows:

$$O(N_{sig}^2 \times |S|^2 \times (\sum_{j \in N_{sig}} t_j) \times R) \tag{6.5}$$

Based on the equations 6.1, 6.2, 6.3, and 6.5, it can be verified that the overall time complexity of our proposed algorithm based on the simple string model and using the exact p-values is equal to

$$O(|S|^2 \times \sum_{i \in \mathcal{W}} t_i + Clog(C) + N_{sig}^2 \times |S|^2 \times (\sum_{j \in N_{sig}} t_j) \times R) \tag{6.6}$$

and in the case of the approximated p-values using the *Poisson* distribution, it is equal to

$$O(|S| \times \sum_{i \in \mathcal{W}} t_i + Clog(C) + N_{sig}^2 \times |S|^2 \times (\sum_{j \in N_{sig}} t_j) \times R) \tag{6.7}$$

6.1.2 Mismatch String Model

In case of the mismatch string model, the time complexity analysis is similar to the simple string model, except in the *Statistical Test Analysis* and the *Core Pattern Set Construction* steps. Again, we go through the analysis step by step:

- **Statistical Test Analysis**: For a mismatch pattern q with maximum number of mismatches d_i, length k_i, and the observed frequency t_i, the worst-case time complexity of computing the probability of occurrence is equal to

$$O(\binom{k_i}{d_i}|\Lambda|^{d_i} \times |k_i|) \tag{6.8}$$

  where $|\Lambda|$ represents the size of the alphabet. Assuming an observed frequency t_i for pattern q_i, and using the *Poisson* approximation, the complexity of the p-value computation for all the extracted patterns can be specified as follows:

$$O(|S| \times \sum_{i \in \mathcal{W}} (t_i \times \binom{k_i}{d_i}|\Lambda|^{d_i} \times |k_i|)) \tag{6.9}$$

  As it was discussed, it is reasonable to assume that the length of the patterns are constant with respect to the length of the sequence. Therefore, the specified time complexity can be simplified as follows:

$$O(|S| \times \sum_{i \in \mathcal{W}} (t_i \times \binom{k_i}{d_i}|\Lambda|^{d_i})) \tag{6.10}$$

- **Correction for Multiple Testing**: The complexity of this step is similar to the case of a simple string model:

$$O(Clog(C)) \tag{6.11}$$

  where C represents the number of patterns whose p-value is less than the significance level α.

- **Core Pattern Set Construction**: As it was discussed in Section 5.3, the explain relation in the case of the mismatch string model is computed using the *Poisson Binomial* distribution. Based on the equations 5.8 and 5.10, computing the p-value of a mismatch pattern q_i of length k_i with observed frequency t_i and the maximum number of mismatches d_i in the presence of a constraint set involves filling the cells of a matrix of dimensions $|S| \times t_i$ using dynamic programming, in addition to the point probabilities which should

be computed for each index in the data sequence S. Therefore, the time complexity of probability computation for each pattern p_i is equal to

$$O(|S| \times |t_i| + |S| \times (\binom{k_i}{d_i} |\Lambda|^{d_i} \times |k_i|)) = O(|S|(|t_i| + (\binom{k_i}{d_i} |\Lambda|^{d_i} \times |k_i|))) \quad (6.12)$$

Given that the *explain* relation should be tested for each pattern against any other pattern in the first step of the greedy search (for simplicity and for the worst-case complexity analysis we ignore the explain relations that can be discarded based on the Theorem 18), the time complexity of the first step of the search algorithm is equal to

$$O(N_{sig} \times |S| \sum_{j \in N_{sig}} (|t_j| + (\binom{k_j}{d_j} |\Lambda|^{d_j} \times |k_j|))) \quad (6.13)$$

where N_{sig} is the number of significant patterns passed to the greedy search algorithm. Based on the assumption that the number of core patterns returned by the algorithm is equal to R, the greedy search is repeated R times. Therefore, the worst-case time complexity of the core pattern set construction can be specified as follows:

$$O(N_{sig} \times |S| \sum_{j \in N_{sig}} (|t_j| + (\binom{k_j}{d_j} |\Lambda|^{d_j} \times |k_j|)) \times R) \quad (6.14)$$

Using equations 6.10, 6.11, and 6.14, the overall worst-case time complexity of our proposed framework based on the mismatch string model can be specified as follows:

$$O(|S| \times \sum_i (t_i \times \binom{k_i}{d_i} |\Lambda|^{d_i}) \quad + \quad (6.15)$$

$$Clog(C) \quad +$$

$$N_{sig} \times |S| \sum_{j \in N_{sig}} (|t_j| + (\binom{k_j}{d_j} |\Lambda|^{d_j} \times |k_j|)) \times R) \quad (6.16)$$

Chapter 7

Experimental Evaluation

To evaluate the performance of our method, we ran experiments on both real and synthetic data. Our synthetic datasets are constructed by *implanting* (i.e. *inserting*) motifs of different lengths in background sequences that are generated using a Markov model of order 1. In order to control the amount of deviations of motifs from the background distribution, a transition probability matrix is used to describe the background distribution and another transition probability matrix that differs from the background, is used to model an anomaly.

For real data, we use three publicly-available datasets; the first one is a log of Unix commands executed by a group of users [94]. This dataset is used for detecting masqueraders (*i.e.* people who use somebody else's computer account without authorization). A training and a test dataset is provided for each user U [94]. The training data contains the commands executed by the user U. The test data consists of commands executed by user U seeded with masquerading users (*i.e.* with commands from other users). The commands executed by each user represent the usage pattern of that user while commands seeded from other users represent different usage patterns, and can be considered as anomalies. The second dataset is an ECG recording [45], showing the electrical impulses of a heart during electrocardiogram tests, and we want to detect anomalies in the form of *arrhythmias*. The third dataset is a motif discovery benchmark that is widely used for evaluating motif finding algorithms [109], and we use it to evaluate our extended model for approximate pattern matching.

For the experiments with the Synthetic data, the ECG data, and the Masquerading users data, we run our pattern discovery method based on the simple model (with exact matching definition of patterns). In all the experiments, the p-values are calculated using the Poisson approximation technique presented in formula C.5, except in the experiments on the synthetic datasets, in which exact p-values are calculated. In these experiments, the *Matthews Correlation Coefficient* (MCC) [76] is used as a measure of matching between the true positions of anomalies in the data with those positions predicted by our test methods. This measurement has been widely used for measuring two-class prediction tasks [9]. The MCC metric is defined as:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(FP + FN)(TN + FP)(TN + FN)}} \tag{7.1}$$

In order to calculate the MCC, the predictions are converted to a binary vector

in which every position that overlaps with a detected anomaly is marked as 1 and every other position is marked as 0.

For experiments on the Synthetic data, the exact p-values are calculated, based on the techniques presented in Section 4.1. In all other experiments, the p-values are approximated using the *Poisson* distribution, as it is presented in more details in Section C.2.1 (Appendix C).

7.1 Evaluation Partners

In order to choose proper comparison partners for our method in finding significant patterns of unknown length in a sequence, two main criteria should be considered. First, a comparison method should be able to detect surprising subsequences (such as local anomalies) of different lengths in a long sequence. Second, a comparison method should be general enough to detect surprising patterns based on their distribution, instead of relying on domain-specific properties of the surprising patterns.

We compared the performance of our method (based on the simple exact matching model) with the five best methods from a comparative evaluation study performed by Chandola *et al.* [20]. Our competitors include the KNN, t-STIDE (STIDE), FSA, FSAz, and HMM, which are summarized in Table 7.1. We chose these methods as our evaluation partners because these methods meet the above-mentioned comparison criteria and have been used in a similar comparative study of anomaly detection techniques for sequence data (the implementations of the competitors were provided as part of this comparative study by Chandola *et al.* [20]).

All of these methods require a length parameter, which is the size of the *window* used for scanning a test sequence and estimating the probability of the subsequence within that window. The KNN method requires an additional parameter K for the K^{th} nearest neighbour. Also, all of these methods generate a probability vector; the vector represents for each position i the probability of occurrence of a subsequence of specific length starting at index i of the test sequence. In order to derive a binary vector representing the positions of anomalies, an additional *probability threshold* parameter is needed. For a given threshold α, the positions in the probability vector with values higher than α are labelled as 0 (*i.e.* normal) and other positions are labelled as 1 (*i.e.* anomalies).

Table 7.1: Summary of Comparison Partners

Method	**Principle**	**Details**
KNN [20]	Similarity-based Anomaly Detection, which is based on computing the distance to closest 'matches' in the training data.	A model is trained using normal sequences in a training phase. In the test phase, each test sequence is compared against the trained model and the closest 'distance' (or K^{th} nearest distance, in general) with the model is considered as the *anomaly score* of the test sequence. The *likelihood score* is the inverse of the *anomaly score*.
Cont'...		

Table 7.1 –

Method	Principle	Details
t-STIDE [112]	Window-based Anomaly Detection, which is based on assigning a likelihood score to each window based on its expected frequency observed in the training data.	A normal profile is created from a dataset of normal sequences by extracting all windows of a fixed length w. This normal profile captures the expected frequencies of subsequences in the training data. In the test phase, all subsequences of length w are extracted from each test sequence and a 'likelihood score' is assigned to each window based on its expected frequency in the training data.
FSA [77]	*Markovian* Anomaly Detection, the probability of observing each symbol a in the test sequence is conditioned on a limited number of symbols preceding the symbol a.	In training, subsequences of length $n+1$ are extracted using sliding window, and a Finite State Automata (FSA) is constructed in which every node corresponds to a unique subsequence of n symbols that form the first n symbols of such $n+1$ length subsequences. Local values are stored in each node which denote the number of observed subsequences corresponding to this node and also the transitions from the current node to other nodes based on the training data. During test, the FSA is used to determine a likelihood score for every subsequence q of length $n+1$ which is extracted from a test sequence S_{test}. This score is equal to the conditional probability associated with the transition from the state corresponding to first n symbols of q to the state corresponding to the last n symbols of q. If there is no state in the automaton corresponding to the first n symbols, the subsequence is ignored.

Cont'...

Table 7.1 –

Method	Principle	Details
FSAz [20]	*Markovian* Anomaly Detection	A variant of the *FSA*, in which if there is no state corresponding to the first n symbols of a subsequence of length $n+1$, a low score is assigned to that subsequence, instead of ignoring it.
HMM [112]	*Hidden Markov Model-based* Anomaly Detection, in which the anomalies are discovered by deviations from the normal state transitions and emission probabilities of an HMM	The training phase involves learning an HMM with hidden states, from the normal sequences using the *Baum-Welch* algorithm. During the testing phase, subsequences of a specific length are extracted using a sliding window technique, and the likelihood score of each subsequnece is computed using the *forward* algorithm.

7.2 Experiments on the Synthetic Data

Our synthetic data is generated using a Markov model of order 1, where we first generate a transition probability matrix that describes the background and another transition probability matrix that differs from the background, modelling an anomaly. The transition probability matrix has most of its transition probability mass in the diagonal, describing the behaviour of a random walk process where with high probability the walk stays in the same state and with small probability it moves to a different state. To capture this intuition, we use the following conditions in generating the transition probabilities out of a state s_i: (1) $P(s_i|s_i) > \sum_{i \neq j} P(s_j|s_i)$, i.e. it is more likely to stay in the same sate rather than changing the state, and (2) $\frac{P(s_j|s_i)}{P(s_k|s_i)} = \frac{|j-i|}{|k-i|}$, i.e. the chance of a move from a state s_i to any other state is proportional to its distance from s_i, assuming that the states are ordered based on their indices.

We ran several experiments, keeping the background distribution fixed and varying the distributions of the anomalies. The self-transition probability is defined as

$$P(s_i|s_i) = \frac{1 + \beta(n-1)}{n}$$

where n is the number of states of the Markov model (which is also the number of symbols we use for generating sequences) and $0 \leq \beta \leq 1$. We set β at 0.9 for the background distribution and used the values 0.7, 0.5, 0.3 and 0.1 of β for the distributions of the anomalies, resulting in increasing amount of deviation from the background distribution. This gave us a range of anomaly models, each shaving some probability mass from the diagonal and spreading it to the rest of the transition matrix. For each experiment, we generated a sequence of a fixed length (set at 400) using the background distribution and implanted shorter sequences that were generated by the anomaly models. In one experiment, referred to as *ExpVarLen*,

the implanted sequences in each test sequence varied in length from 4 to 10, whereas the frequency and the number of unique subsequneces were set to the fixed value 4. In another experiment, *ExpFixLen*, the implanted sequences in an individual test sequence were of the same length. Separate experiments are performed for each value of length in the range [4 : 10], where the frequency and number of unique subsequences were set to 4, and the anomaly distributions were varying according to different values of β. The main difference between these two experiments is that in the experiment *ExpVarLen* the implanted motifs in a test sequence are of different lengths. However, in the experiment *ExpFixLen* all motifs in a test sequence are of the same length. Another key difference between these two experiments is that in the experiment *ExpVarLen* the lengths of implanted patterns are unknown to all methods; However. in the experiment *ExpFixLen* our competitors know the true lengths of patterns while our method does not know (and not require) it.

We generated a sequence using our background process for training. For evaluation, independent sequences with embedded anomalies as described above were used. We generated 10 datasets under each setting and measured the average performance of methods on these 10 datasets.

The values of the parameters for the comparison methods usually are not known in a real setting and our method does not require these parameters. To compare with methods that require such parameters, we explore a reasonable part of the parameter space and report their 'best' and 'average' performances. To that end, we ran these methods for all lengths in the range [4 : 10] in *ExpVarLen*. In *ExpFixLen*, we set the window length parameter of our competitors to the true length of implanted anomalies which is fixed in *ExpFixLen*. We also ran the KNN method for all values of K in the range [1 : 10]. As a *probability threshold*, we considered all the distinct values in the probability vector that is generated by a competitor, and used these threshold values for reporting the 'best' and the 'average' performance of our competitors. We chose the threshold which results in the highest MCC value for reporting the 'best' performance of our competitors. The 'average' performance of competitors are evaluated in two ways.

1. **avg_threshold**: In this mode, we compute the average of MCCs on all the threshold values which were explored.

2. **heu_threshold**: In this mode, we use a heuristic to choose a 'suitable' threshold among all the threshold values, and compute the MCC based on the estimated threshold. In this heuristic, referred to as *MaxGap*, the probability values of a sequence are sorted in the ascending order and the differences between consecutive values are calculated. The estimated threshold is the mean of two consecutive probability values which results in the largest difference. The intuition is that a suitable threshold should provide a larger gap between the probability values corresponding to normal and anomalous positions compared to those between two normal probability values or two anomalous probability values.

In the *ExpVarLen* experiments, we also used another method for computing the 'average' performance of our competitors, in addition to the 'average' performance measuring methods presented above. In this method, which we will refer to as the

avg_len_best_threshold, the probability threshold value (and similarly the value of K in the case of the KNN method) which results in the highest MCC for each window length is selected, and the final MCC is averaged over all window lengths. The purpose of this setting is to evaluate the sensitivity of our competitors solely on the parameter length while the other parameters are selected in a way which results in the highest MCC (for a particular window length). This method for computing 'average' performance is not applicable in the *ExpFixLen* experiments because the value of the parameter length is known to each competitor.

The results of the *ExpVarLen* and *ExpFixLen* experiments are shown in Figures 7.1(a)/7.1(b) and 7.4(a)/7.4(b), respectively. All plots show the result on 4 different sequences with different anomaly models (decreasing values for β, corresponding to decreasing deviations of the anomalies from the background model). Figures 7.1(a) and 7.4(a) show the results when providing the comparison methods with the 'best' parameter settings (which would not be known in real settings), and Figures 7.1(b) and 7.4(b) compare our results with the 'average' performance of each comparison method in which the probability threshold values are computed using the *heu_threshold* heuristic (which represents a more realistic *expected* performance). One can clearly see that our method outperforms all the other methods, even when giving them the "unfair" advantage of providing the 'best' parameter setting.

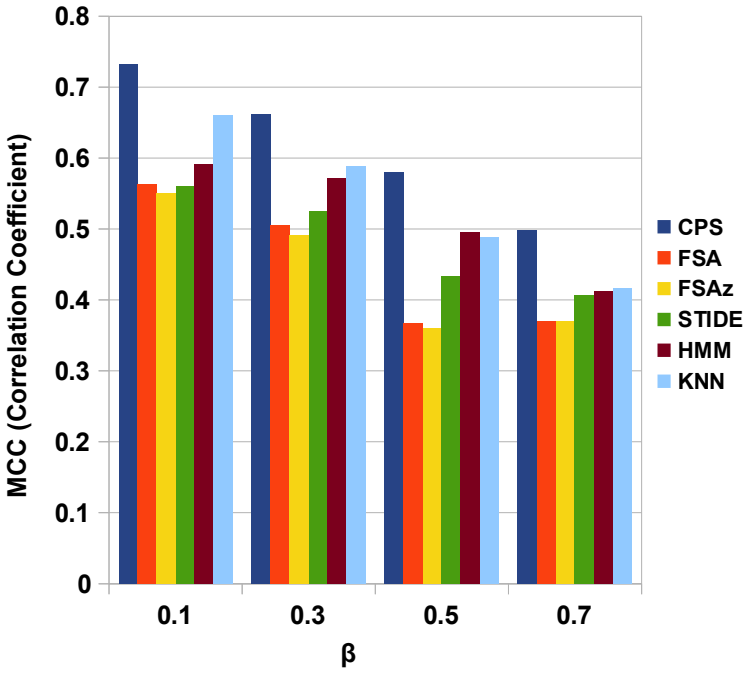
Comparing with the expected performance, our method dominates the comparison methods to an even larger extent. Figures 7.2 and 7.5 compare our results with the 'average' performance of each comparison method in which the probability threshold values are computed using the *avg_threshold* heuristic. Similar results can be observed on these comparisons on domination of our method over the comparison partners when another method is used for computing the average performance for our competitors. Moreover, it shows that the *heu_threshold* method delivers relatively better results compared to those of the *avg_threshold* heuristic in most of the settings.

The results achieved by the CPS, when compared with the average performance of other partners computed using the *avg_len_best_threshold* method in the Figure 7.3, show that our method perform better than the average results of our competitors when the best probability threshold (and the best of value of the parameter K in the case of the KNN method) are used. This can be considered as a comparison between our method and our competitors solely on the parameter *length*, in settings in which other parameters except the value of the length are set to the 'best' possible value for our competitors.

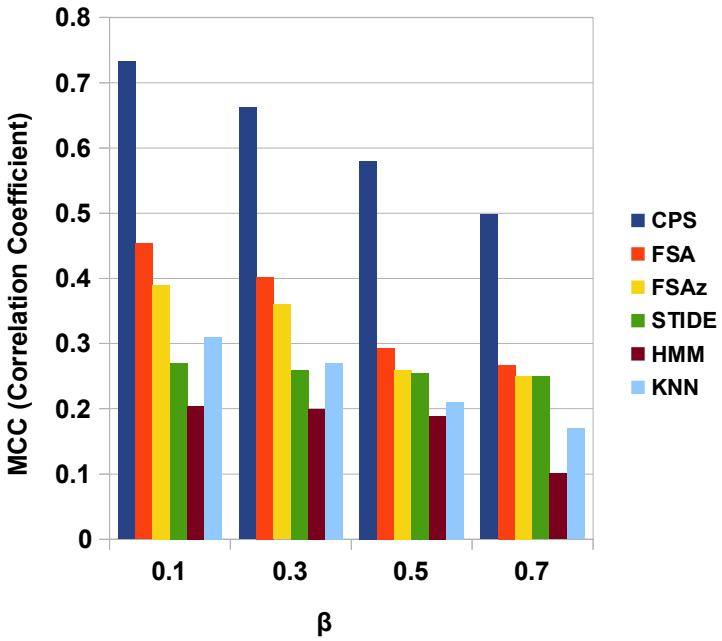
The results of the experiments also demonstrate that as the deviation between the background model and anomaly models increases (*i.e.* as the β value decreases), it becomes easier to detect anomalies, and as a result most of the methods perform better for smaller values of β.

Given that in synthetic experiments 1 ($ExpVarLen$) and 2 ($ExpFixLen$) 10 datasets is generated for each settings of parameters, we measure the significance of difference between the results of our proposed method (CPS) and our competitors using *paired-sample t-Test*.

The results of paired-sample t-Tests between CPS and other competitors on synthetic experiments 1 and 2 are shown in Tables 7.2 and 7.3, respectively. The



(a) CPS vs. best performance of other methods.



(b) CPS vs. average performance of other methods computed in the *heu_threshold* mode.

Figure 7.1: Performance comparison between methods in *ExpVarLen*, varying the β of anomaly models.

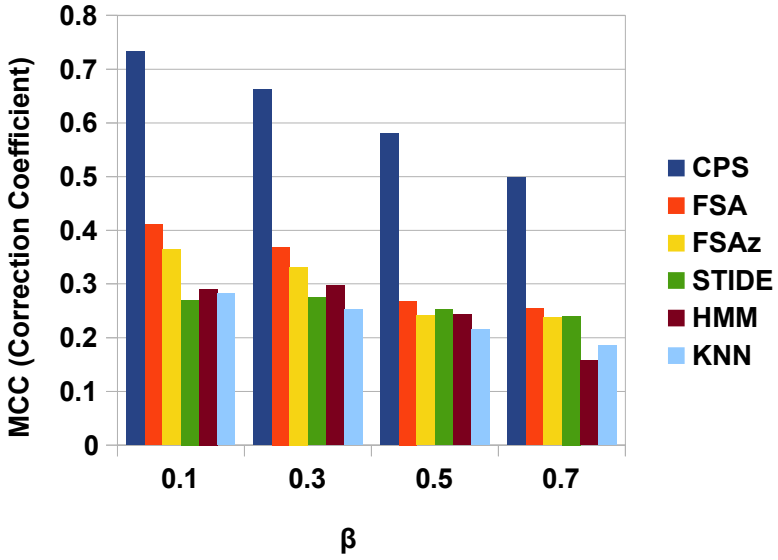


Figure 7.2: Performance comparison between methods in *ExpVarLen*, varying the β of anomaly models. CPS vs. performance of other methods computed in the *avg_threshold* mode (*i.e.* MCC averaged over different window lengths and different probability threshold values and also the K parameter for the *KNN* method).

first column in each table shows the pairs of methods on which the test is performed. We consider the 'best' and 'average' performance of each competitor separately. The second column in each table represents the p-value, that is the probability that the null hypothesis is rejected by chance. At a significance level $\alpha = 0.01$, the null hypotheses are rejected for all the paired t-Tests, in which the p-values range between $3.6136e-21$ and 0.001, and are generally much smaller than the significance level 0.01. The weakest results belong to the paired-sample t-Test between CPS and KNN_{best} in both of the synthetic experiments, whereas the strongest results belong to the comparison of our method with KNN_{avg} and HMM_{avg} in the synthetic experiments 1 and 2, respectively.

7.3 Experiments on the ECG Dataset

Anomalies in ECG data can be observed within individual heartbeats or in a sequence of heartbeats. If abnormalities persist over consecutive heartbeats, an *arrhythmia* occurs. Examples of heartbeat types include the *Normal*(N), *Atrial premature* (A), *Premature Venctricular contraction*(V) and *missed* (M) beats. *Atrial bigeminy*(AB) and Ventricular trigeminy (T) are examples of arrhythmias. For our experiments, we used the MIT-BIH Arrhythmia dataset [80] which contains 48 half-hour excerpts of ECG recordings, obtained from 47 subjects.

We randomly selected 4 patient records, and for each record, we used part of the data that does not include any arrhythmia for training. The test data consists

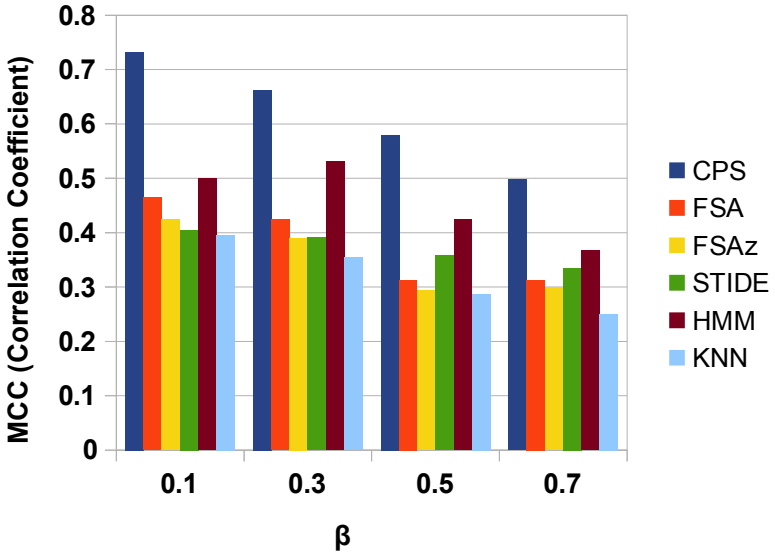
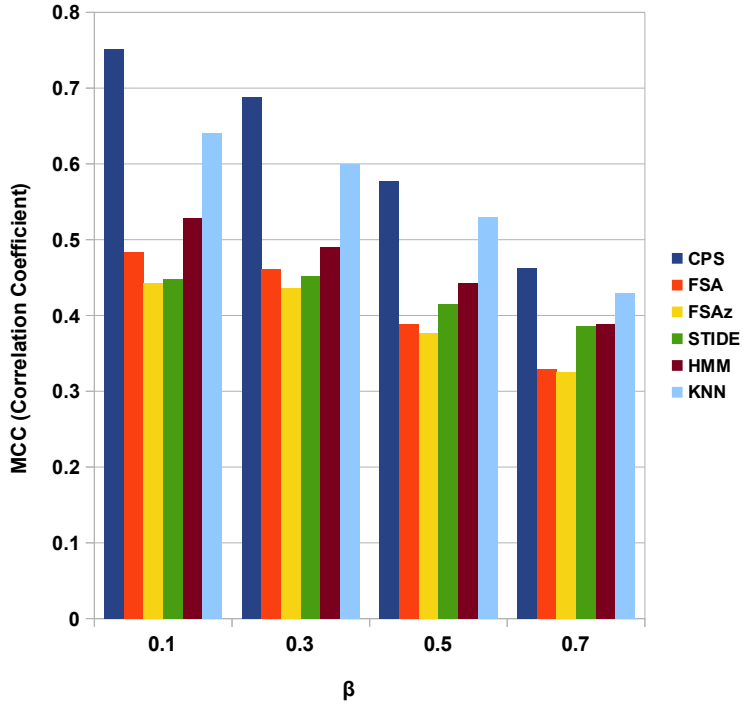
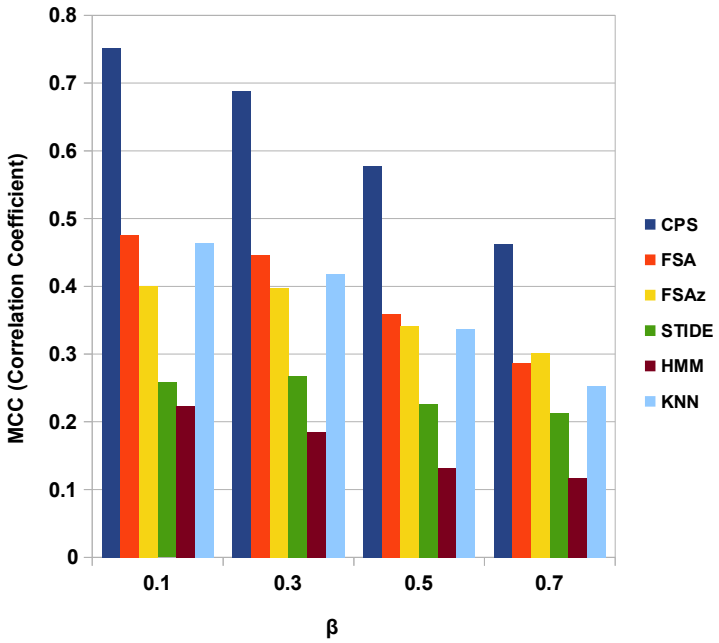


Figure 7.3: Performance comparison between methods in *ExpVarLen*, varying the β of anomaly models. CPS vs. performance of other methods computed in the *avg_len_best_threshold* mode (*i.e.* MCC averaged over different window lengths while the best probability threshold values and the best K for the KNN method are selected for each length.)



(a) CPS vs. best performance of other methods.



(b) CPS vs. average performance of other methods computed in the *heu_threshold* mode.

Figure 7.4: Performance comparison between methods in *ExpFixLen*, varying the β of anomaly models.

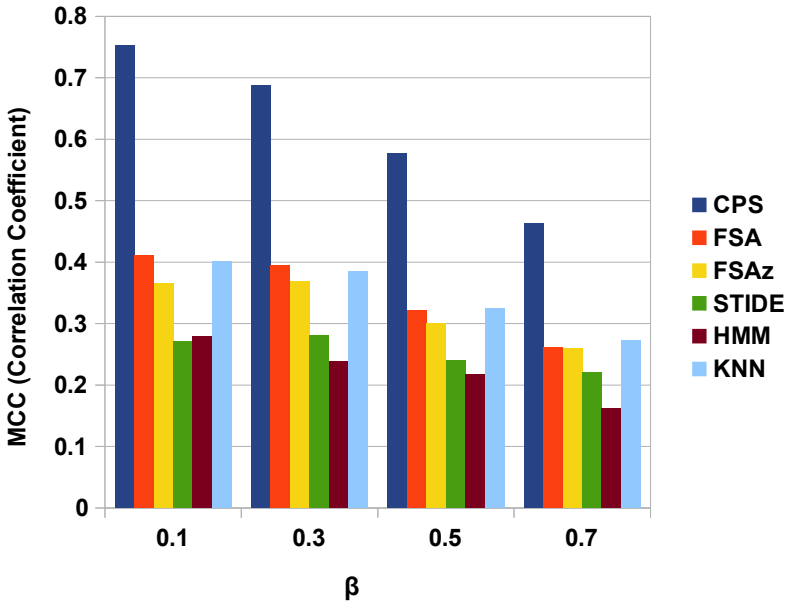


Figure 7.5: Performance comparison between methods in *ExpFixLen*, varying the β of anomaly models. CPS vs. performance of other methods computed in the *avg_threshold* mode (*i.e.* MCC averaged over different window lengths and different probability threshold values and also the K parameter for the *KNN* method).

Table 7.2: Paired-sample t-Test for Synthetic Experiment $ExpVarLen$ (significance level $\alpha = 0.01$)

Pairs	**p-value**
(CPS, FSA_{best})	8.3052e-10
(CPS, FSA_{avg})	2.1530e-15
(CPS, $FSAz_{best}$)	9.0129e-11
(CPS, $FSAz_{avg}$)	4.1352e-17
(CPS, $STIDE_{best}$)	4.5743e-09
(CPS, $STIDE_{avg}$)	2.6041e-15
(CPS, HMM_{best})	1.5723e-04
(CPS, HMM_{avg})	5.7493e-17
(CPS, KNN_{best})	4.6182e-05
(CPS, KNN_{avg})	3.6136e-21

Table 7.3: Paired-sample t-Test for Synthetic Experiment $ExpFixLen$ (significance level $\alpha = 0.01$)

Pairs	**p-value**
(CPS, FSA_{best})	1.2742e-07
(CPS, FSA_{avg})	8.3487e-10
(CPS, $FSAz_{best}$)	7.7841e-08
(CPS, $FSAz_{avg}$)	1.9875e-09
(CPS, $STIDE_{best}$)	5.4365e-06
(CPS, $STIDE_{avg}$)	2.0451e-12
(CPS, HMM_{best})	5.4365e-06
(CPS, HMM_{avg})	6.3542e-15
(CPS, KNN_{best})	0.0015
(CPS, KNN_{avg})	9.7907e-10

of both normal and arrhythmia intervals. For our competitors, the parameter space was explored as follows. For each dataset, a competitor was run for all window values in the range $[2 : 10]$. For the KNN method, the experiments were also run for all KNN values in the range $[1 : 10]$ for any given length. We compared the performance of our method again with the 'best' and 'average' performance of our competitors, based on the same definitions of the 'best' and 'average' results given in Subsection 7.2.

Figure 7.6(a) compares the results of our method and the 'best' possible performance of our competitors on records 231, 201, 124, and 223, in which the possible probability thresholds are explored using the Ths_{all} method. The results reveal that our method outperforms or performs close to the best results achieved by other methods when giving them the advantage of knowing best parameter settings. When comparing to a more realistic expected performance of our competitors in Figure 7.6(b), we can observe that our method achieves significantly higher MCCs in all cases, except for the record 231, on which most methods perform well.

Table 7.4 shows the patterns found by our method (column 2) and true arrhythmias in the data (column 3). The results show that the notion of a core pattern set

in our method closely matches with the definition of arrhythmias. For instance, the pattern *VNNV* in record 201 precisely corresponds to the definition of a "T" (Ventricular trigeminy) arrhythmia, which happens when two normal beats are observed between two V beats. Also, the pattern *VVV* in record 124 precisely specifies the definition of an "IVR" (Idioventricular rhythm) arrhythmia, which happens when three (or more) V beats are observed consecutively. An interesting observation is that the patterns found by our method capture the core part of an arrhythmia, though the arrhythmia might last for a longer time interval. An example is the "BII" arrhythmia that might last for a long interval of the form MRMRM..., MNMNM..., or a combination of both. This arrhythmia is represented in our method by two concise patterns MRM and MNM, capturing the core parts of this arrhythmia.

Table 7.4: Discovered patterns vs. true Arrhythmia in MIT-BIH records

Rec	Discovered patterns	True arrhythmias
201	**1)**VNNV **2)** jNjAj	**1)** T : VNNVNNV... **2)** NOD: jNjAj
223	**1)** VVV **2)** VNV	**1)** IVR : VVV... **2)** B : NVNVNV... **3)** T : VNNVNNV...
124	**1)**VVV **2)** NNV	**1)** IVR : VVV... **2)** T : VNNVNNV...
231	**1)**MRM **2)** MNM	**1)** BII : MRMRMRM... **2)** BII : MNMNMN **3)** BII : MRMRM...MNMNMN...

7.4 Experiments on the Masquerading User Dataset

The masquerading dataset consists of commands for 50 users. For each user, 15,000 commands are collected, of which the first 5,000 commands do not contain any masquerader commands. The next 10,000 commands might contain commands from other users (*i.e.* masqueradors) in blocks of size 100. Commands from other users are inserted in the test dataset so that only a small portion of the data is anomalous [94]. To use this dataset for detecting anomalies of variable lengths, we generate a smaller dataset by sampling data from the masquerading user dataset so that anomalous sequences of different lengths exist in the data and also the properties of the original dataset are maintained. To achieve this, the test sequence for each user is generated from the provided test sequence as follows: the size of seeded anomalous subsequences is chosen to be distributed uniformly in the range [3:10]; the size of normal subsequences are distributed uniformly in the range [20:90], and the number of anomalous subsequences in each test sequence is distributed uniformly in the range [2:5]. We used the original training sequence, unchanged, for the purpose of training.

The results of our experiments are shown in Figures 7.7(a) and 7.7(b). The 'best' performance corresponds to the highest MCC achieved by a competitor when different values of the *length* and *probability threshold* parameters are explored (as well as the parameter K, in case of KNN). In the reported 'best' case performance in Figure 7.7(a), the probability thresholds are explored using the Ths_{all} method. For the 'average' performance, the reported MCC is the average MCC value for a competitor over all possible values of the length parameter (and possible values of K, in case of KNN); the *probability threshold* is estimated using the *MaxGap* heuristic. As the results of the experiments show, our method clearly dominates

the average performance of other competitors in most of the cases, except one in which our performance is still a close second. In the case when the competitors are given the unrealistic advantage of using the best parameter setting, our method performs still better (in different degrees) in 5 out of the 8 cases; in the remaining 3 cases some of the other methods produced slightly better results. For this set of experiments, we could not report the results for the HMM-based method because the model did not converge after a long time.

7.5 Evaluating the Extended Model: An Application to the Motif Discovery Problem

In this section, we evaluate the performance of the extended model in comparison to a motif discovery benchmark that is widely used for evaluating motif finding algorithms [109]. The goal of a motif discovery method is to identify regulatory elements, notably the *binding sites* in DNA sequences, for *transcription factors*. In molecular biology and genetics, a transcription factor is a protein that binds to specific DNA sequences, thereby controlling the flow (or *transcription*) of genetic information.

Little is known about most *transcription factors* and their target *binding sites*. Computational tools have been developed for discovering novel motifs, where nothing is assumed a priori of the transcription factor or its preferred binding sites. Co-regulated genes are known to share some similarities in their regulatory mechanism, possibly at transcriptional level, so their promoter regions might contain some common motifs that are binding sites for transcription factors. Thus, a way to detect these regulatory elements is to search for statistically overrepresented motifs in the promoter region of such set of co-expressed genes. Automatic identification of these motifs is a challenging task. The binding sites of a single transcription factor have different lengths and sequence codings, in the presence of a great amount of noise.

7.5.1 Benchmark Explanation

For the evaluation of our method (with approximate pattern matching model), a well-known dataset is used that was presented by Tompa *et al.* in order to assess the performance of 14 motif discovery tools [109]. This dataset has become a major benchmark for the motif discovery problem and has been widely used in the literature. The datasets in this benchmark were generated based on real transcription factors contained in the TRANSFAC database [3]. Each selected real transcription factor from the TRANSFAC database gives rise to three datasets. These datasets are the same with respect to the binding sites they contain (and the positions of the binding sites), but differ in the type of the background sequences containing the binding sites. The background sequences are of three types: 1) '*real*', which are selected from binding sites' real promoter sequences, 2) '*generic*', which are based on randomly chosen promoter sequences from the same genome, and 3) 'markov', which are generated randomly from a Markov chain of order 3. The transcription factors in this benchmark are selected from 4 species, including *yeast*, *fly*, *mouse*, and *human*.

The resulting benchmark contains 56 datasets, including 6 datasets from *fly*, 26 from *human*, 12 from *mouse* and 8 from *yeast*. As negative control, 4 datasets of type *markov* (background sequence) contain no motifs. The number of sequences per dataset varies between 1 and 35 with mean 7, the individual sequence length per dataset varies from 500 bp (base pairs) to 3,000 bp. The total size of each dataset varies from 1 to 70 kb with mean 8 kb. The number of implanted binding sites per dataset varies from 0 to 76 with mean 9.

The dataset is available online at the assessment website [1]. For each dataset, a prediction tool is supposed to select the single *best motif* and report the positions of that motif's binding sites, or to report that the dataset contains no significant motif.

We ran our proposed method for finding generalized surprising patterns on this benchmark. To learn the background distribution, we used the regulatory sequence tools available in the RSAT database (http://rsat.ulb.ac.be/rsat/) [106]. For each species, the first 1000 base pairs of genome sequences were used as input in order to learn the parameters of a Markov chain model of order 3.

7.5.2 Evaluation Metrics

In order to compare the computational tools, 7 metrics are proposed by Tompa *et al.* [109] which consider the matching of predictions and true binding sites both in the level of *nucleotides* and *sites*. The proposed metrics include the *Sensitivity*, *Positive predictive value*, *Performance coefficient*, *correlation coefficient*, and *Average site performance*.

At the finest granularity level, the matching between predicted binding sites and the known (true) binding sites can be measured at the level of nucleotides. According to [109], the following raw measures are defined at the nucleotide level:

- **nTP**: the number of nucleotide positions that are in both predicted sites and known sites
- **nFN**: the number of nucleotide positions in known sites that are not in predicted sites
- **nFP**: the number of nucleotide positions in predicted sites not in known sites
- **nTN**: the number of nucleotide positions in neither known sites nor predicted sites

The metrics can also be defined at the level of sites rather than nucleotides. According to Tompa *et al.* [109], a *predicted* site 'overlaps' a *known* site if they overlap by at least one-quarter of the length of the known site. Then, the corresponding raw statistics at the site level can be defined as follows [109]:

- **sTP**: the number of known sits that are 'overlapped' by predicted sites
- **sFN**: the number of known sites that have no 'overlap' with predicted sites
- **sFP**: the number of predicted sites not 'overlapped' by known sites

Based on the nucleotide-level and site-level statistics defined above, the following measures can be defined at the site level ($x = s$) or nucleotide level ($x = n$) according to [109]:

1. **Sensitivity**: defines the fraction of known sites or nucleotides that are predicted:

$$\text{Sensitivity: xSn} = \frac{xTP}{xTP + xFN} \tag{7.2}$$

2. **Positive Predictive Value**: defines the fraction of predicted sites or nucleotides that are true predictions (known) :

$$\text{Positive Predictive Value: xPPV} = \frac{xTP}{xTP + xFP} \tag{7.3}$$

3. **Specificity**: defines the fraction of negative nucleotide positions (non-motif positions) which are correctly predicted as such:

$$\text{Specificty: nSP} = \frac{nTN}{nTN + nFP} \tag{7.4}$$

4. **Performance Coefficient**

$$\text{Performance Coefficient: nPC} = \frac{nTP}{nTP + nFN + nFP} \tag{7.5}$$

5. **Correlation Coefficient**: which is the Pearson correlation coefficient in the particular case of binary classification, in which the predicted nucleotide positions and the known nucleotide positions are considered as binary sequences of 1s and 0s, and this statistic measures the correlation between these 1s and 0's. The value of nCC ranges between -1 (indicating perfect anti-correlation) and $+1$ (indicating perfect correlation). This measure is also referred to as the *Matthews Correlation Coefficient*, as defined in the equation 7.6.

$$\text{Correlation Coefficient (nCC)} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(FP+FN)(TN+FP)(TN+FN)}} \tag{7.6}$$

6. **Average Site Performance**

$$\text{Average Site Performance : sASP} = \frac{sSn + sPPV}{2} \tag{7.7}$$

The above-mentioned statistics are undefined in case an algorithm predicts no motif. In this case, $TP + FP = 0$, and therefore the measures *nCC*, *nPPV*, *sPPV*, and *sASP* are undefined. Moreover, a technique is required to aggregate the statistics over a collection of datasets. For instance, we might need to aggregate a statistic such as *nCC* over all datasets which belong to a particular species such as *yeast*, or we might be interested to have an aggregated results for a statistics over all datasets with Markov background distribution. To address these issues, the following method is used by Tompa *et al.* for evaluating the methods on the proposed

benchmark [109]. For a computation tool T, statistic M, and each collection of C of datasets, the *combined* measure of each of the 7 statistics defined above are calculated as follows. The raw statistics nTP, nFP, nFN, nTN, sTP, sFP, and sFN are added up for all datasets in C as if C is a long sequence containing the sequences of all datasets contained in it. The values of measures calculated using aggregated statistics are referred to as the *combined* measures. Using the combined measures, the probability of getting an undefined value for a measure is reduced greatly. The reason is that the undefined value happens only if a motif finding algorithm does not predict a motif in any of the datasets in a collection C, which is unlikely to happen.

7.5.3 Evaluation Partners on Motif Discovery

The tools compared in Tompa's assessment include AlignACE [55], ANN-Spec [117], Consensus [51], GLAM [38], Improbizer [5], MEME [8], MITRA [31], MotifSampler [105], oligio/dyad-analysis [110], QuickScore [88], SeSiMCMC [33], Weeder [85], and YMF [97]. A short description of each method is provided in Table 7.5.

Table 7.5: Summary of the comparison partners used in the motif discovery experiments (adapted from table 1 in [109])

Algorithm	**Principle**	**Details**
AlignACE [55]	Gibbs sampling algorithm that returns a series of motifs as weight matrices that are over-represented in the input set	Judges alignments sampled during the course of the algorithm using a *maximum a priori* log likelihood score, which gauges the degree of overrepresentation. Provides an adjunct measure (group specificity score) that takes into account the sequence of the entire genome and highlights those motifs found preferentially in association with the genes under consideration
ANN-Spec [117]	Models the DNA-binding specificity of a transcription factor using a weight matrix	Objective function based on log likelihood that transcription factor binds at least once in each sequence of the positive training data compared with the number of times it is estimated to bind in the background training data. Parameter fitting is accomplished with a gradient descent method, which includes Gibbs sampling of the positive training examples
Consensus [51]	Models motifs using weight matrices, searching for the matrix with maximum information content	Uses a greedy method, first finding the pair of sequences that share the motif with greatest information content, then finding the third sequence that can be added to the motif resulting in greatest information content, and so on.
Cont'...		

Table 7.5 –

Algorithm	Principle	Details
GLAM [38]	Gibbs sampling-based algorithm that automatically optimizes the alignment width and evaluates the statistical significance of its output	Since the basic algorithm cannot find multiple motif instances per sequence, long sequences were fragmented into shorter ones, and the alignment was transformed into a weight matrix and used to scan the sequences to obtain the final site predictions
Improbizer [5]	Uses expectation maximization to determine weight matrices of DNA motifs that occur improbably often in the input sequences	As a background (null) model it uses up to a second-order Markov model of background sequence. Optionally, Improbizer constructs a Gaussian model of motif placement, so that motifs that occur in similar positions in the input sequences are more likely to be found.
MEME [8]	Optimizes the E-value of a statistic related to the information content of the motif	Rather than sum of information content of each motif column, statistic used is the product of the P-values of column information contents. The motif search consists of performing expectation maximization from starting points derived from each subsequence occurring in the input sequences. MEME differs from MEME3 mainly in using a correction factor to improve the accuracy of the objective function.
MITRA [31]	Uses an efficient data structure to traverse the space of IUPAC patterns.	For each pattern, MITRA computes the hypergeometric score of the occurrences in the target sequences relative to the back-ground sequences and reports the highest scoring patterns.
MotifSampler [105]	Matrix-based, motif-finding algorithm that extends Gibbs sampling by modeling the background with a higher order Markov model	To improve the robustness of the Gibbs sampling algorithm on noisy data sets, this algorithm is extended for motif finding with a higher-order background model, and it is shown that the use of a higher-order model considerably enhances the performance.
oligio/dyad [110]	Detects overrepresented oligonucleotides with oligo-analysis 15 and spaced motifs with dyad-analysis	These algorithms detect statistically significant motifs by counting the number of occurrences of each word or dyad and comparing these with expectation. The Most crucial parameter is the choice of the appropriate probabilistic model for the estimation of occurrence significance. In this study, a negative binomial distribution on word distributions was obtained from 1,000 random promoter selections of the same size as the test sets.

Cont'...

Table 7.5 –

Algorithm	**Principle**	**Details**
QuickScore [88]	Based on an exhaustive searching algorithm that estimates probabilities of rare or frequent words in genomic texts	Incorporates an extended consensus method allowing well-defined mismatches and uses mathematical expressions for efficiently computing z-scores and P-values, depending on the statistical models used in their range of applicability. Special attention is paid to the drawbacks of numerical instability. The background model is *Markovian*, with order up to 3.
SeSiMCMC [33]	Modification of Gibbs sampler algorithm that models the motif as a weight matrix, optionally with the symmetry of a palindrome or of a direct repeat, and optionally with spacer	Includes two alternating stages. The first one optimizes the weight matrix for a given motif and spacer length. The algorithm changes the positions of the motif occurrences in the sequences and infers the motif model from the current occurrences. These changes are used to optimize the likelihood of sequences as being segmented into the (Bernoulli) background and the motif occurrences. The optimization is organized via a Gibbs-like Markov chain, which samples positions in sequences one by one, until the Markov chain converges. The second stage looks for best motif and spacer lengths for obtained motif positions. It optimizes the common information content of motif and of distributions of motif occurrence positions.

Cont'...

Table 7.5 –

Algorithm	Principle	Details
Weeder [85]	Consensus-based method that enumerates exhaustively all the oligos up to a maximum length and collects their occurrences (with substitutions) from input sequences	Each motif is evaluated according to the number of sequences in which it appears and how well conserved it is in each sequence, with respect to expected values derived from the oligo frequency analysis of all the available upstream sequences of the same organism. Different combinations of canonical motif parameters derived from the analysis of known instances of yeast transcription factor binding sites (length ranging from 6 to 12, number of substitutions from 1 to 4) are automatically tried by the algorithm in different runs. It also analyzes and compares the top-scoring motifs of each run with a simple clustering method to detect which ones could be more likely to correspond to transcription factor binding sites. Best instances of each motif are selected from sequences using a weight matrix built with sites found by a consensus-based algorithm.
YMF [97]	Uses an exhaustive search algorithm to find motifs with the greatest z-scores	A P-value for the z-score is used to assess significance of motifs. Motifs themselves are short sequences over the IUPAC alphabet, with spacers (Ns) constrained to occur in the middle of the sequence.

To cope with the constraints defined in this competition and to deal with domain-specific characteristics of patterns for the motif discovery problem, some modifications are necessary in our general framework for finding *surprising patterns*. The modifications are performed in different steps of our methodology, including multiple test correction, selecting a single *best motif* from a core pattern set, and reporting the final binding sites for a *core motif*. These are discussed in the following subsections.

7.5.4 Multiple Test Correction Using False Discovery Rate

As mentioned in Section 4.2, we use in our general framework for pattern discovery the *Holm* procedure [53] to correct for *multiple hypothesis testing*. The Holm procedure is an example of a 'family-wise error rate' (FWER) procedures, which take a conservative approach in selecting significant observations in favour of reducing false positives (*Type I errors*). In contrast, the *False Discovery Rate* (FDR) procedures take a less conservative approach in identifying the important few significant observations from thousands of null hypothesis tested.

In the following experiments on the motif discovery benchmark, we used the Storey procedure [101] with threshold 0.05 on the q-values of the patterns to deter-

mine the patterns for which the null hypothesis is rejected. The selected patterns are passed as input to the *core pattern set* selection algorithm.

7.5.5 Selecting a single best motif from a core pattern set

In our general framework for pattern discovery, the result of our algorithm is a *core pattern set* which might consist of several patterns. In the benchmark chosen for evaluation, only a single motif (*transcription factor*) is expected to be returned. To meet this condition, we select the pattern that is found in the first step of the greedy search algorithm 4.2 (*i.e.* the pattern which *explains* the highest number of other significant patterns) as the single *best motif*.

7.5.6 Reporting binding sites from a core pattern

In order to evaluate a motif discovery method on the given benchmark, the *binding sites* and their locations in the input sequences should be reported as final predictions. The *binding sites* are *instances* of the discovered motif which exist in the data based on the defined *matching* criteria. When a mismatch pattern $Q_{s,d}$ is selected as the top pattern in a core pattern set by our algorithm, the binding sites can be reported as all instances of Q which appear in the input sequences (i.e. all subsequences of length $|s|$ in the data sequences with *edit distance* less than d *w.r.t* string s). This approach leads to some binding sites being reported, which are *false positives*. To mitigate this problem, we excluded binding sites that look *dissimilar* to the majority of the other binding sites. To characterize this property, we used a concept similar to a *Position Weight Matrix* (PWM) model [102] that was simplified to a *frequency matrix* by Pavesi *et al.* [85].

The pruning procedure using a *frequency matrix* can be summarized as follows. For a mismatch pattern $Q_{s,d}$, a frequency matrix M is of size $4 \times |s|$, and initialized with 0 values in all cells. The matrix has 4 rows, one for each of the 4 nucleotides A, C, G and T in a DNA sequence. Each cell $m_{i,j}$ in a frequency matrix represents the number of times a nucleotide x, corresponding to the i^{th} row of the matrix, appears in the j^{th} position of an occurrence of a *mismatch pattern*. The matrix can be derived by scanning all *occurrences* (binding sites) of the pattern Q in the data sequence and updating the corresponding cells of the matrix. For each *hit* (occurrence) h of the pattern Q, and for each character h_j in this hit, let i represent the row number which matches with character h_j. Then, the corresponding cell of the matrix M is updated as $M_{i,j} = M_{i,j} + 1$. Then, for each occurrence $h = h_1, h_2, ..., h_{|s|}$ a *score* is computed by matching this occurrence against the frequency matrix M:

$$Z(h) = \sum_{j=1}^{|s|} log(m_{i,j}) \tag{7.8}$$

where $m_{i,j}$ is the entry of the matrix corresponding to the j^{th} nucleotide of h. The maximum and minimum scores in M, represented by $Z_{max}(M)$ and $Z_{min}(M)$, are also computed by summing up the maximum and minimum entries of each column of M in the logarithm scale. The score of each occurrence h can be then scaled to a percentage value between 0 and 100 using the following formula:

$$Z^{'}(h) = \frac{100 * (Z(h) - Z_{min}(M))}{Z_{max}(M) - Z_{min}(M)} \tag{7.9}$$

This matching score gives us a means to prune all the occurrences (binding sites) with matching scores less than a threshold. We used a threshold value of 90% for pruning.

Figure 7.9 compares the overall performance of *CPS* (our proposed core pattern set method) with all 14 competitors on the given benchmark. The overall performance has been computed on all the 56 datasets. The reported metric is the *correlation coefficient* measure defined in equation 7.6. The *combined nCC* in Figure 7.9 stands for the correlation coefficient on a single sequence created by appending all the sequences in the 56 datasets. The results of the comparison between the *CPS* algorithm with other computational tools based on different measures (introduced by Tompa *et al.* in their comparative study [109]) is provided in Figure 7.8.

As it is shown in Figure 7.9, our proposed method achieves a performance close to Weeder [85], which performs the best among the other competitors overall. The detailed performance of each method on different species (*Fly*, *Human*, *Mouse*, and *Yeast*) is shown in figure 7.10. It can be observed that the performance of most motif discovery methods is significantly better on the yeast datasets. The *CPS* algorithm achieves the best results compared to all other methods on the *mouse* datasets. Also, it achieves comparable results to Weeder on the *human* and *fly* datasets.

The performance of the methods separated based on different background sequence types (*Real*, *Markov*, *Generic*) is shown in Figure 7.11. The CPS algorithm stand as the second best method on *Generic* background data, and the third best on *Markov* background data. All the methods perform very poor on datasets with *real* background sequences, on which the *CPS* algorithm achieves a performance close to best results.

Even though motif discovery is a challenging problem and still an active topic of research, the low measure of combined correlation in the reported results should not be considered as inefficiency of computational methods for motif discovery as discussed by providers of the benchmark [109]. One of the reasons is that each method is supposed to return the best motif in each dataset (or none). This makes the selection process a very challenging task. In practice, it might be more effective to investigate the top motifs rather than the single *best* motif. As we will discuss in the following, this setting will increase the prediction power of a motif discovery tool.

Another feature of the benchmark which makes prediction harder is that many of the binding sites cataloged in the TRANSFAC database are usually long (*e.g.* 35 of the binding sites used in the benchmark dataset are 31-71 bp in length). This makes binding site modelling more challenging due to the fact that variety between binding sites generally increases with their lengths. Also, the datasets with *real* background sequences might not be *fully* annotated, and the existence of other transcription factor binding sites is possible in these datasets. This might lead to an unfair evaluation of some methods which discover the *unknown* binding sites (*i.e.* the binding sites that have not been annotated in the TRANSFAC database).

In practice, it might be useful for a motif discovery method to return a list of top K motifs based on an internal *likelihood score*, which can later be verified by domain

experts. This could potentially increase the number of 'true' binding sites which can be found by a motif discovery method, regardless of *false positives*. To pursue this scenario, we designed a new experiment using the same benchmark where we evaluated the performance of the 'best' motif among the top 32 motifs returned by a discovery tool. The comparison metric is the same *combined correlation coefficient* used in previous experiments on this benchmark. As comparison partner, we selected *Weeder* [85] which achieved the best performance among the competitors on this benchmark.

The Weeder algorithm works by computing the 'significance score' for a class of motifs. Each class of motif is defined by the length of the string it represents (6, 8, 10, 12), the number of mismatches allowed (depends on the length of the motif and takes a value between 1 and 4), and a threshold on the minimum percentage of DNA sequences that a pattern should occur in, to be considered as a candidate motif (50%, 100%). This gives rise to 16 classes of motifs in total. The Weeder algorithm computes a 'significance score' for a motif m, which roughly compares the observed frequencies of m in the test sequences with their 'expected' frequencies (This score captures a notion of the 'over-representation' of motifs, but is different from the concrete p-value measure that is used in our proposed CPS method). Five motifs with highest scores are selected in each class, and a single *best* motif among them is selected through some heuristics. These heuristics try to select a motif which has the highest score in its class, and is both *vertically-redundant* and *horizontally-redundant* (The details of these heuristics can be found in the parameters document submitted for each competitor along with binding sites in the assessment webiste [1]). Then, the binding sites of the *best* selected motif are selected and pruned by computing their score against a *frequency matrix* model as in Equation 7.9. The source code for the main step of finding top motifs in different classes has been made available by the authors. We used the Weeder program to return the top 2 motifs in each class, giving rise to a total of 32 motifs which are used in the top-K experimental evaluation.

The result of the comparison between *Weeder* and *CPS* is shown in Figure 7.12. As can be observed in this figure, our proposed *CPS* algorithm performs better than Weeder on all types of species, except on the human datasets in which both techniques achieve comparable results. We observed that the performance of Weeder in the top-K experiment is weaker than in the 'best' motif selection settings used in the original competition. In an effort to investigate the issue, we implemented the Weeder's heuristics in a Matlab script. Despite all the efforts, we could not reproduce the reported results for Weeder algorithm in the original 'best motif' experiments on this benchmark. However, the candidate motifs used in the top-K experimental settings are solely derived from the Weeder source code (made available by the authors), and the results of comparison with the CPS algorithm demonstrate the superiority of our proposed algorithm compared to Weeder, when used in a top-K motif evaluation settings.

In previous experiments, we evaluated the performance of methods solely based on the *nucleotide-level Correlation Coefficient* (nCC) because it provides a comparison at a fine granularity between predicted binding sites and *true* binding sites. We can look at other measures to evaluate the performance of a motif discovery tool in terms of its ability to match (even partially) with true binding sites regardless

of false positives. One of these measures that is introduced and used by Tompa *et al.* in their comparison study is *Site-level Sensitivity* (sSn) [109], which is defined as follows:

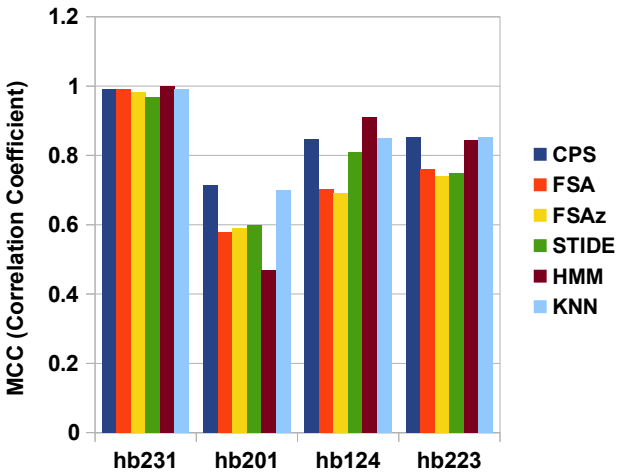
$$\text{Site-level Sensitivity} = \frac{sTP}{sTP + sFN} \tag{7.10}$$

in which the site-level true positive, and site-level false negative are defined as follows:

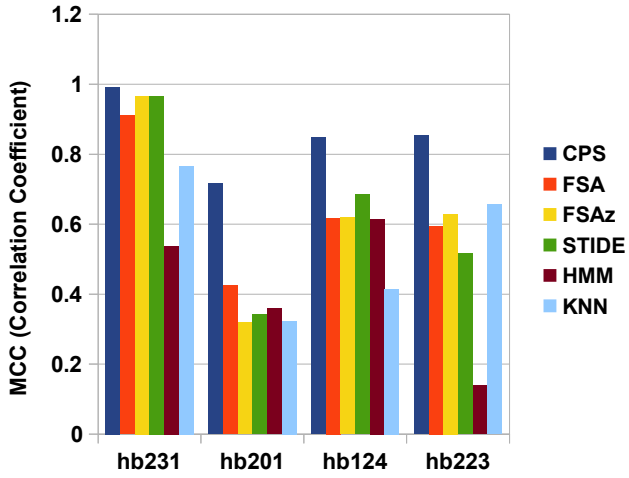
- sTP: the number of known sits that are overlapping with predicted sites
- sFN: the number of known sites that have no overlap predicted sites

By definition, a predicted site overlaps a known site if they overlap by at least one-quarter the length of the known site [109].

Figure 7.13 shows the performance of the top-32 motifs chosen by the CPS and Weeder algorithms in terms of their *site-level sensitivity*. The results show that despite a very low measure on the nucleotide-level correlation coefficient, these two motif discovery methods perform fairly good in (partial) matching with *true binding sites*. The achieved statistics show that we are able to *partially* match with around half of the *true binding sites* of transcription factors if we just look at the top 32 motifs returned by these two methods. It is worth mentioning that even in this comparison we just consider the performance of the *best* motif among the top 32 motif returned by each method and the aggregated *recall* over all the top 32 motifs might be even higher than 50%. The performance of our proposed CPS algorithm is comparable to that of Weeder, which performs the best among the methods used in Tompa's comparative study [109].

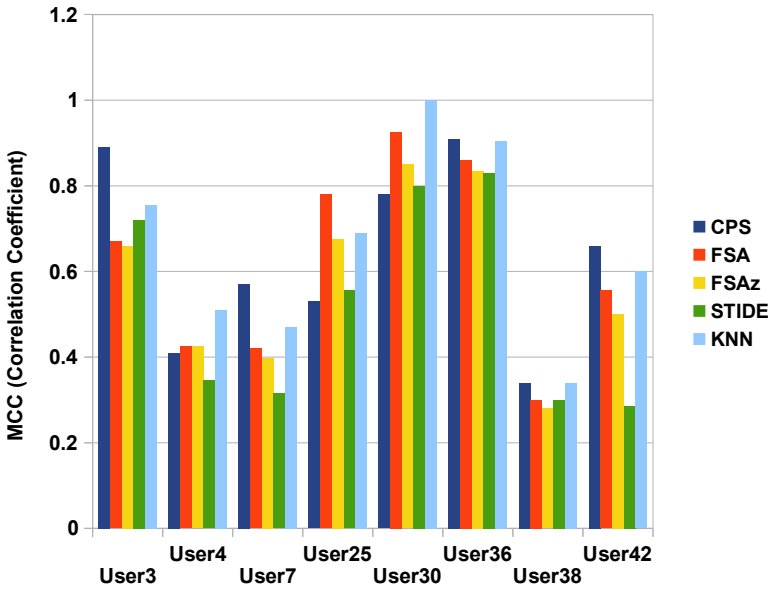


(a) CPS vs. best performance of other methods.

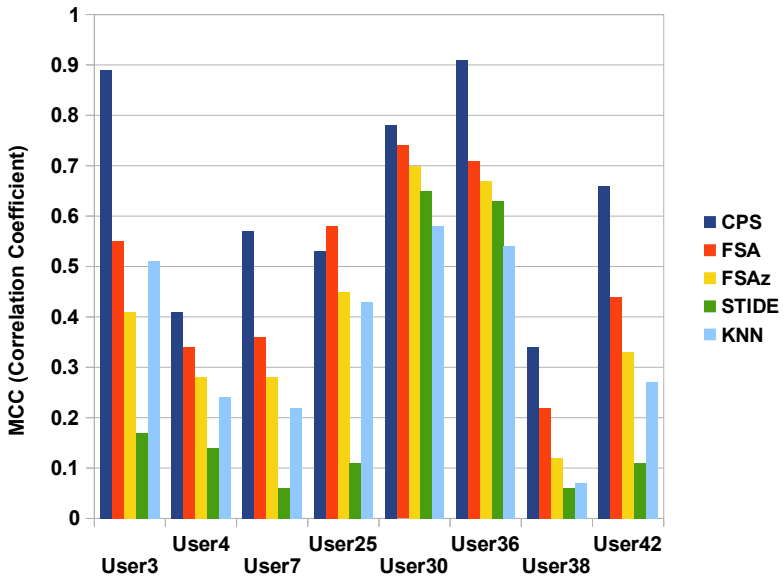


(b) CPS vs. average performance of other methods computed in the *heu_threshold* mod.

Figure 7.6: Performance comparison between methods on ECG dataset.



(a) CPS vs. best performance of other methods.



(b) CPS vs. average performance of other methods.

Figure 7.7: Performance comparison on Masquerading User data computed in the *heu_threshold* mode.

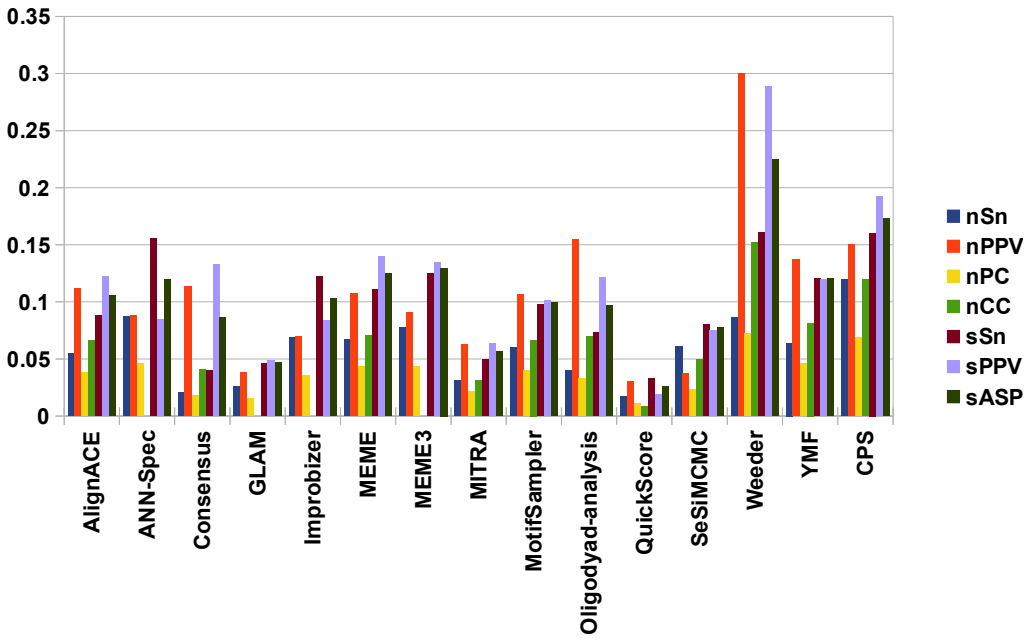


Figure 7.8: Combined measures of correctness over all 56 datasets, based on the evaluation metrics defined by Tompa *et el.* in [109], including the *nSn* (nucleotide-level *Sensitivity*), *nPPV* (nucleotide-level *Positive predictive value*), *nPC* (nucleotide-level*Performance coefficient*), *nCC* (nucleotide-level *correlation coefficient*), *sSn* (site-level Sensitivity), *sPPV* (site-level *Positive predictive value*), and *sASP* (site-level *Average site performance*).

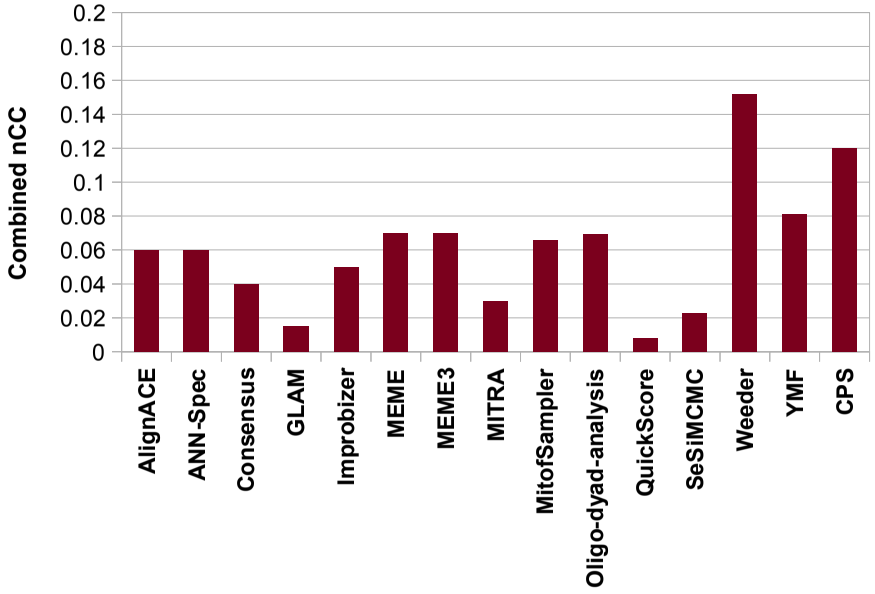


Figure 7.9: *Combined correlation coefficient (nCC) over all 56 datasets.*

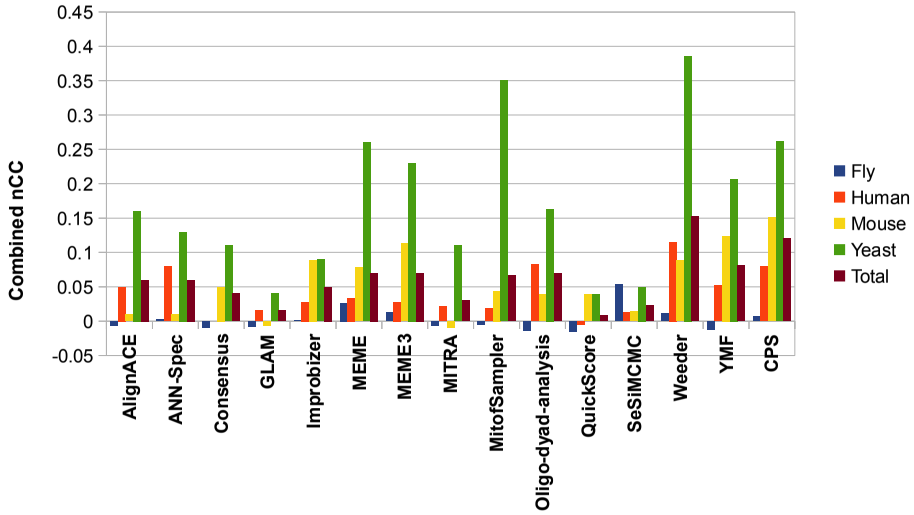


Figure 7.10: *Combined correlation coefficient (nCC) by different species.*

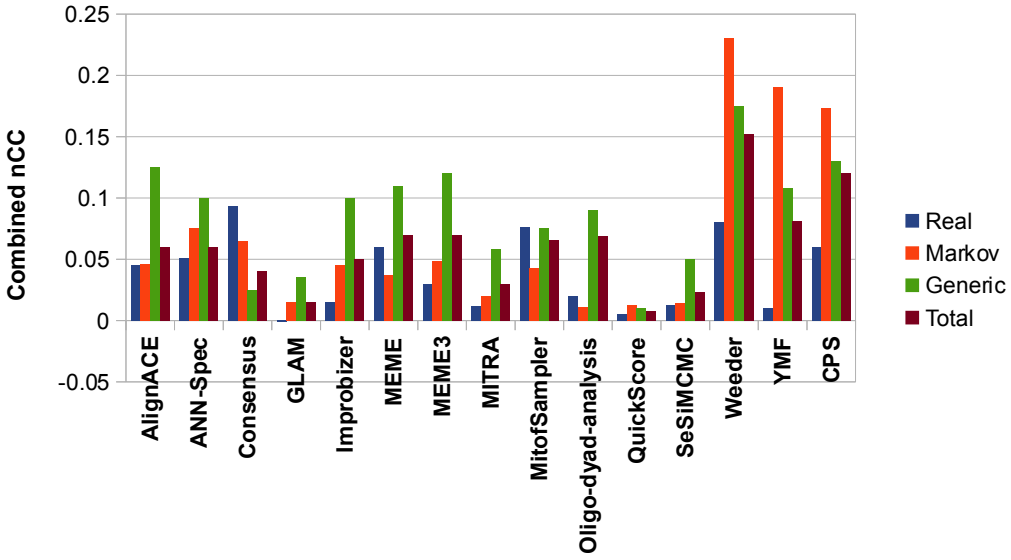


Figure 7.11: *Combined correlation coefficient (nCC) by different backround sequences.*

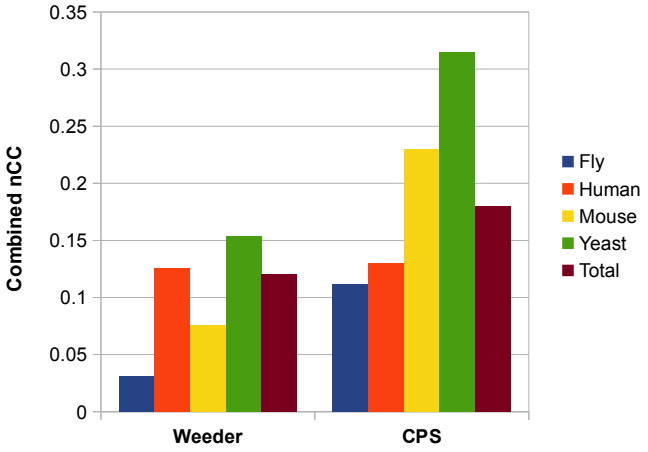


Figure 7.12: *Combined correlation coefficient (nCC) for CPS and Weeder on top 32 chosen motifs.*

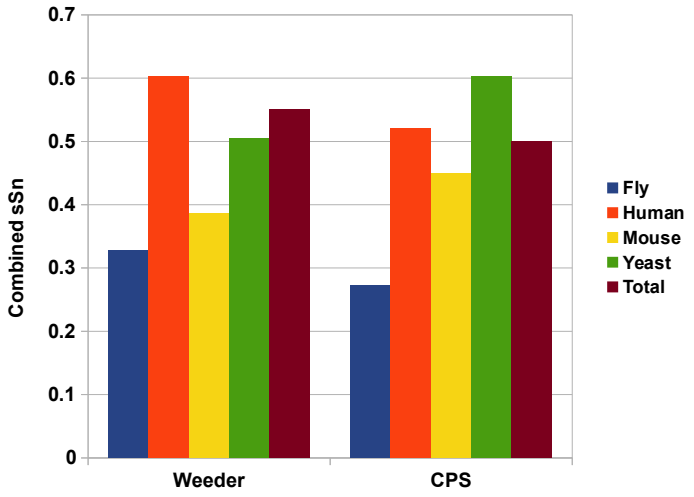


Figure 7.13: *Combined site-level sensitivity (sSn) for CPS and Weeder on top 32 chosen motifs.*

Chapter 8

Conclusions and Future Work

8.1 Research Summary

We investigated the problem of finding surprising patterns in sequences without knowing the lengths of the patterns. The notion of 'surprise' was formalized by the statistical significance of observed frequency of a pattern compared to its 'expected' frequency. We demonstrated the drawbacks of doing multiple statistical tests on subsequences of all possible lengths, which is an alternative when the lengths of 'true' surprising patterns are unknown. We showed that multiple tests on subsequences of all possible lengths will result in a set of highly correlated patterns, from which it is difficult to extract the 'true' surprising patterns. In the light of this observation, we developed statistical methods to capture the dependencies between patterns through an "explain" relationship. These methods not only allow us to justify the correlation between patterns, but also provide basic tools to construct a redundancy-free set of surprising patterns explaining the frequency of all other patterns in the data. Moreover, we developed a new method to compute the p-values of pattern frequencies in sequence data under the new settings, in which specific constraints are imposed at some positions in the data sequence.

The *core pattern set* is introduced as our solution to the formalized problem, which is the smallest set of patterns that can 'explain' the frequency of any other pattern in the sequence. The problem of constructing an exact core pattern set was shown to be NP-complete. Therefore, we proposed a greedy search algorithm to construct an approximate core pattern set. We extended the definition of patterns from a simple string model to a *mismatch string* model which allows some variations in instances which belong to a pattern (*motif*). The algorithms for finding correlations between significant patterns and finding a *core pattern set* were extended to the problem of finding these approximate patterns. This extension provides support for a broader class of applications in which an approximate representation of surprising patterns are required, including the interesting problem of finding transcription factor binding sites in DNA sequences.

We used several strategies to improve the running time of our proposed method. Some approximation and early abandoning techniques were developed to speed-up p-value calculation of patterns, which is a computationally expensive operation in our proposed method. We also characterized the necessary conditions for correlation between two patterns, which gave us a means to prune the expensive explain relation

tests for patterns which are not correlated. The prefix tree data structure was employed to speedup the probability computations for patterns, and reusing the calculations that have been done for prefixes of patterns. We also re-implemented the proposed method in a parallel architecture, which gave us a significant speed-up in the running time of our method by parallel implementation of time consuming tasks of p-value computations and 'explain relation' tests.

The proposed method was evaluated using both synthetic and real-world datasets in the fields of Medicine, Computer Security, and Bioinformatics, for solving interesting problems such as characterizing arrhythmia in ECG recordings of heart patients, finding 'masqueraders' in user command sequences, and identifying transcription factor binding sites (*i.e.* 'motifs') in DNA sequences.

Our proposed method based on the simple string model was compared against 5 well-known anomaly detection methods to evaluate the performance of our method in detecting the anomalous patterns in the synthetic data generated under different settings, which were controlled by the length of *implanted* motifs and the degree of deviation from the background distribution. Our method outperformed the competitors even though they are given the unrealistic advantage of using the best parameter settings. In a more realistic case, in which the best parameter setting is not known, our proposed method dominated the expected performance of our competitors.

The *core pattern set* algorithm based on the simple string model was also evaluated on the ECG and the Masquerade user datasets for detecting anomalous subsequences. The experimental study demonstrated the effectiveness of our proposed method for characterizing heart arrhythmia in ECG dataset. The discovered *core pattern sets* by our algorithm very well matched with the definitions of arrhythmia. Compared to expected performance of other competitors, the *CPS* algorithm achieved a better matching to the ground truth. Compared to the 'best' performance of other competitors, which was chosen by exploring the space of different parameters and represents a biased setting, our proposed method achieved comparable results in most of the cases. Similar behaviour was observed by our method for finding anomalous patterns in the Masquerade user datasets.

A widely-used motif discovery benchmark was selected to demonstrate the application of our proposed method based on the extended model to the motif discovery problem. Our proposed method was used for the problem of motif discovery with few adjustments, which were required to comply with the constraints defined by the benchmark publishers [109], and also to cope with some of the domain-dependent properties of the significant patterns in the field of Bioiformatics. The evaluation partners were selected from 14 well-known motif discovery tools which submitted their results to the competition based on the benchmark. The results demonstrated that our method achieves a performance which is better or comparable to the Weeder motif finding method, which performed the best among the other motif finding tools in a published performance evaluation on the same benchmark [109]. This result is promising considering the fact that our general framework for finding suprising patterns could easily be extended to the challenging problem of biological motif discovery, and was able to achieve results that were comparable to or better than results of some of the well-known methods in this domain.

8.2 Directions for Future Work

The proposed techniques and applications in this research study can be extended in several aspects:

1. The core pattern set approximation can be explored more thoroughly. One interesting issue to investigate is to explore other strategies for breaking the ties in the search algorithm. Currently, we are choosing the pattern with shortest length in case several patterns 'explain' an equal number of patterns. Other choices are possible in case of a tie and we have gotten better results in some experiments when the longest pattern is selected. Another interesting avenue of research is the theoretical analysis of the current greedy algorithm to investigate its *approximation factor*, and to understand the conditions under which the greedy algorithm delivers comparable result to that of an optimum solution. Also, it is interesting to investigate search strategies other than the greedy search algorithm to approximate a core pattern set solution.

2. The number of significant patterns which are passed as input to the core patter set algorithm might be large. This involves a substantial number of *explain relation* tests. On the other hand, the *explain relation* test might return a negative answer for a large number of pairs, as we discussed in Section C.3. Investigating the condition(s) which can limit the scope of *explain relation* tests, especially in case the patterns are allowed to have mismatches sounds challenging and interesting. Perhaps clustering the set of significant patterns could limit the scope of 'explain' relationship tests to a cluster (as opposed to the set of all remaining patterns), and reduce the number of required p-value calculations as a result.

3. A major computationally intensive component of our framework is to calculate p-values for all 'candidate' patterns, which might be a massive set in some domains such as Bioinformatics. A useful research study is to investigate how the intensive p-value computation for a pattern Q, or a group of patterns, can be reduced based on the outcomes of calculations done for some other patterns which are 'correlated' with new patterns in question (*i.e.* have partial overlaps with each other).

4. The framework in this research study gives us a 'template' for finding unexpectedly frequent patterns, and we studied two sample instantiations of this template using the *basic* string and *mismatch* string models, and evaluated them on sample datasets mainly for the problems of anomaly detection and Biological motif discovery. Investigating other instantiations of the template can introduce new avenues of research. Of particular interest to the domain of Bioinformatics is exploring other models for motif representation. For instance, probabilistic models such as 'PWM's can be used instead of the mismatch string model. The challenges that are still to be addressed are the problems of setting a threshold for a PWM and characterising the *explain* relation between PWM's.

5. In making efforts to apply our proposed framework to the problem of motif discovery, we tried to use a general mismatch string to allow variations in a

pattern, without leveraging other domain-specific characteristics of patterns. However, Biologically-motivated properties of motifs can be captured to make the patterns more specific and guide the pattern enumeration process. Biological motifs are known to have specific structures. For instance, a large number of motifs are known to be the *palindromic*. A *palindromic* motif is a subsequence that is exactly the same as its own reverse complement, *e.g.*, "CACGTG". A *Spaced Dyad* motif is another common type of motif which consists of two smaller conserved sites, separated by a *spacer* (*i.e.* gap). The spacer occurs in the middle of the motif because the transcription factors bind as a dimer. This means that the transcription factor is made out of two subunits that have two separate contact points with the DNA sequence. The parts where the transcription factor binds to the DNA are conserved but are typically rather small (3-5 bp). An interesting extension of our work is to investigate how incorporating these structures in our motif definitions can limit the search space or decrease the false positives by ignoring the binding sites which don't match the common structures.

6. Pattern enumeration techniques for motif discovery suffer from the false positive problem. In Section 7.5.6, we used a simple method based on the frequency matrix score and an arbitrary threshold 90% to prune some of the binding sites which are less similar to other binding sites, and decrease the false positives. Investigating other approaches to reducing false positives is of great benefit to a pattern-based motif finding algorithm. Biology-motivated constraints such as palindrome property could reduce the number of false positives. More systematic approaches have been proposed in the literature [50], which are based on identifying all the potential matching sites using a pattern-based enumeration, converting these binding sites to a PWM, and then deriving a threshold for the PWM based on the *order statistics*. A direct extension of our proposed method is to study how other alternatives such as Hartmann's method [50] can be used as a post-processing step in our methodology to make the motif more 'specific', and improve the matching of reported binding sites to the known motifs.

Bibliography

[1] Assessment of computational motif discovery tools. `http://bio.cs.washington.edu/assessment/`.

[2] *Abbreviations and Symbols for Nucleic Acids, Polynucleotides and Their Constituents: Recommendations 1970*. International Union of Pure and Applied Chemistry, 1970.

[3] TRANSFAC: A Database on Transcription Factors and Their DNA Binding Sites. *Nucleic Acids Research*, 24(1):238–241, January 1996.

[4] R. Agrawal and R. Srikant. Mining sequential patterns. In *IEEE International Conference on Data Engineering*, pages 3–14, 1995.

[5] W. Ao, J. Gaudet, W. J. Kent, S. Muttumu, and S. E. Mango. Environmentally induced foregut remodeling by pha-4/foxa and daf-12/nhr. *Science*, 305(5691):1743–1746, 2004.

[6] I. Assent, E. Mller, S. Gnnemann, R. Krieger, and T. Seidl. Less is more: Non-redundant subspace clustering. In *International Workshop on Discovering, Summarizing and Using Multiple Clusterings*, 2010.

[7] T. L. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning*, 21(1-2):51–80, 1995.

[8] T. L. Bailey and C. Elkan. The value of prior knowledge in discovering motifs with MEME. In *International Conference on Intelligent Systems for Molecular Biology*, pages 21–29, 1995.

[9] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424, 2000.

[10] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occuring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[11] Y. Benjamini. Discovering the false discovery rate. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4):405–416, 2010.

[12] Y. Benjamini and Y. Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 57(1):289–300, 1995.

[13] S. Budalakoti, A. Srivastava, R. Akella, and E. Turkov. Anomaly detection in large sets of high-dimensional symbol sequences. *NASA Ames Research Center, NASA TM-2006-214553*, 2006.

[14] S. Budalakoti, A. N. Srivastava, and M. E. Otey. Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 39(1):101–113, 2009.

[15] J. B. D. Cabrera, L. M. Lewis, and R. K. Mehra. Detection and classification of intrusions and faults using sequences of system calls. *SIGMOD Record*, 30(4):25–34, 2001.

[16] N. Castro and P. Azevedo. Time Series Motifs Statistical Significance. In *SIAM International Conference on Data Mining*, pages 687–698, 2011.

[17] S. Chakrabarti, S. Sarawagi, and B. Dom. Mining surprising patterns using temporal description length. In *International Conference on Very Large Data Bases*, pages 606–617, 1998.

[18] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58, July 2009.

[19] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839, 2012.

[20] V. Chandola, V. Mithal, and V. Kumar. Comparative evaluation of anomaly detection techniques for sequence data. In *IEEE Internation Conferenece in Data Mining*, pages 743–748, Washington, 2008.

[21] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Annual Meeting on Association for Computational Linguistics*, pages 310–318, 1996.

[22] B. Chiu, E. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. In *ACM SIGMOD international conference on Management of data*, pages 493–498, 2003.

[23] M. C. Chuah and F. Fu. ECG anomaly detection via time series analysis. In *IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 123–135, 2007.

[24] W. W. Cohen. Fast effective rule induction. In *International Conference on Machine Learning*, pages 115–123, 1995.

[25] P. G. da Fonseca, K. S. Guimarães, and M. F. Sagot. Efficient representation and P-value computation for high-order Markov motifs. *Bioinformatics*, 24(16):160–166, 2008.

[26] E. Davidson, J. Rast, P. Oliveri, A. Ransick, and C. etc. Calestani. A genomic regulatory network for development. *Science*, 295(5560):1669–1678, 2002.

[27] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1999.

[28] V. J. Easton and J. H. McColl. Statistics glossary. `http://www.stats.gla.ac.uk/steps/glossary/time_series.html`.

[29] I. Erill. A gentle introduction to information content in transcription factor binding sites. Technical report, Department of Biological Sciences, University of Maryland Baltimore County (UMBC), 2012.

[30] E. Eskin. Modeling system calls for intrusion detection with dynamic window sizes. In *IEEE DARPA Information Survivabilty Conference and Exposition*, pages 165–175, 2001.

[31] E. Eskin and P. A. Pevzner. Finding composite regulatory patterns in dna sequences. *Bioinformatics*, 18(1):354–363, 2002.

[32] François Fauteux, Mathieu Blanchette, and Martina V. Strömvik. Seeder: discriminative seeding DNA motif discovery. *Bioinformatics*, 24(20):2303–2307, 2008.

[33] A. V. Favorov, M. S. Gelfand, A. V. Gerasimova, D. A. Ravcheev, A. A. Mironov, and V. Makeev. A gibbs sampler for identification of symmetrically structured, spaced dna motifs with improved estimation of the signal length. *Bioinformatics*, 21(10):2240–2245, 2005.

[34] Pedro G. Ferreira, Paulo J. Azevedo, Cndida G. Silva, and Rui M. M. Brito. Mining approximate motifs in time series. In *International Conference on Discovery Science*, pages 7–10, 2006.

[35] A. Floratou, S. Tata, and J. M. Patel. Efficient and Accurate Discovery of Patterns in Sequence Data Sets. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1154–1168, 2011.

[36] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268 – 278, 1973.

[37] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *IEEE Symposium on Security and Privacy*, pages 120–128, 1996.

[38] M. C. Frith, U. Hansen, J. L. Spouge, and Z. Weng. Finding functional sequence elements by multiple local alignment. *Nucleic Acids Research*, 32(1):189–200, 2004.

[39] Liu G., McDaniel T.K., S. Falkow, and S. Karlin. Sequence anomalies in the cag7 gene of the helicobacter pylori pathogenicity island. *Proceedings of the National Academy of Sciences USA*, 96(12):7011–7016, 1999.

[40] W. A. Gale and K. W. Church. What is wrong with adding one? In *Corpus-based Research into Language*, pages 189–198. 1994.

[41] W. A. Gale, K. W. Church, and M. Hill. Estimation procedures for language context: poor estimates are worse than none. In *Computational Statistics*, pages 69–74, 1990.

[42] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[43] A. K. Ghosh, A. Schwartzbard, and M. Schatz. Learning program behavior profiles for intrusion detection. In *Workshop on Intrusion Detection and Network Monitoring*, pages 51–62, 1999.

[44] A. K. Ghosh, A. Schwartzbard, and M. Schatz. Using program behavior profiles for intrusion detection. In *SANS Conference and Workshop on Intrusion Detection and Response*, pages 6–6, 1999.

[45] A. L. Goldberger et al. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):215–220, 2000.

[46] I.J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3):237–264, 1953.

[47] R. Gwadera, M. J. Atallah, and W. Szpankowski. Markov models for identification of significant episodes. In *SIAM International Conference on Data Mining*, pages 404–414, 2005.

[48] R. Gwadera and F. Crestani. Ranking sequential patterns with respect to significance. In *Pacific Asia Knowledge Discovery and Data Mining*, pages 286–299, 2010.

[49] S. Hanhijärvi. Multiple hypothesis testing in pattern discovery. In *International conference on Discovery science*, pages 122–134, 2011.

[50] H Hartmann, E. W. Guthöhrlein, M. Siebert, S. Luehr, and J. Söding. P-value-based regulatory motif discovery using positional weight matrices. *Genome research*, 23(1):181–194, 2013.

[51] G. Z. Hertz and Gary D. Stormo. Identifying dna and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15(7):563–577, 1999.

[52] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Computer Security*, 6(3):151–180, 1998.

[53] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

[54] Y. Hong. On computing the distribution function for the sum of independent and non-identical random indicators. *Computational Statistics & Data Analysis*, 59(2):41–51, 2011.

[55] Jason D. Hughes, Preston W. Estep, Saeed Tavazoie, and George M. Church. Computational identification of Cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae1. *Molecular Biology*, 296(5):1205–1214, March 2000.

[56] A. K Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[57] S. M. Katz. Estimation of probabilities from sparse data for the language mode component of a speech recognizer. 35(3):400–401, 1987.

[58] E. Keogh, J. Lin, and A. Fu. HOT SAX: efficiently finding the most unusual time series subsequence. In *IEEE International Conference on Data Mining*, pages 226–233, 2005.

[59] E. Keogh, S. Lonardi, and B. Y. Chiu. Finding surprising patterns in a time series database in linear time and space. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 550–556, 2002.

[60] E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 206–215, 2004.

[61] J. Kleffe and M. Borodovsky. First and second moment of counts of words in random texts generated by Markov chains. *Computer Applications in the Biosciences*, 8(5):433–441, 1992.

[62] T. Lane. *Machine Learning Techniques for the Computer Security Domain of Anomaly Detection*. PhD thesis, Purdue University, 2000.

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

[64] C. E. Lawrence and A. A. Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, 7(1):41–51, 1990.

[65] W. Lee, S. J. Stolfo, and P. K. Chan. Learning patterns from unix process execution traces for intrusion detection. In *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56, 1997.

[66] C. Li, Q. Yang, J. Wang, and M. Li. Efficient mining of gap-constrained subsequences and its various applications. *ACM Transactins on Knowledge Discovery from Data*, 6(1):1–39, 2012.

[67] Y. Li and J. Lin. Approximate variable-length time series motif discovery using grammar inference. In *Workshop on Multimedia Data Mining*, pages 1–9, 2010.

[68] S. Liang, M. P. Samanta, and B. A. Biegel. Cwinnower algorithm for finding fuzzy DNA motifs. *Bioinformatics and Computational Biology*, 2(1):47–60, 2004.

[69] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 2–11, 2003.

[70] J. Lin, E. J. Keogh, A. W. Fu, and H. V. Herle. Approximations to magic: Finding unusual medical time series. In *IEEE Symposium on Computer-Based Medical Systems*, pages 329–334, 2005.

[71] D. Liu, X. Xiong, B. DasGupta, and H. Zhang. Motif discoveries in unaligned molecular sequences using self-organizing neural networks. *IEEE Transactions on Neural Networks*, 17(4):919–928, 2006.

[72] X. Liu, Douglas L. Brutlag, and Jun S. Liu. Bioprospector: Discovering conserved dna motifs in upstream regulatory regions of co-expressed genes. In *Pacific Symposium on Biocomputing*, pages 127–138, 2001.

[73] X. Liu, Douglas L. Brutlag, and Jun S. Liu. An algorithm for finding protein-dna binding sites with applications to chromatin-immunoprecipitation microarray experiments. *Nat Biotechnol.*, 20(8):835–839, 2002.

[74] C. Low-Cam, C. Rassi, M. Kaytoue, and J. Pei. Mining statistically significant sequential patterns. In *IEEE International Conference on Data Mining*, pages 488–497, 2013.

[75] C. Marceau. Characterizing the behavior of a program using multiple-length N-grams. In *Workshop on New Security Paradigms*, pages 101–110, 2000.

[76] B. W. Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta*, 405(2):442–451, 1975.

[77] C. C. Michael and A. Ghosh. Two state-based approaches to program-based anomaly detection. In *IEEE Annual Computer Security Applications*, pages 203–237, Washington, DC, 2000.

[78] R. G. Miller. *Simultaneous Statistical Inference*. Springer-Verlag, New York, 1991.

[79] G. Moise and J. Sander. Finding non-redundant, statistically significant regions in high dimensional data: a novel approach to projected and subspace clustering. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 533–541, 2008.

[80] G.B. Moody and R.G. Mark. The impact of the MIT-BIH arrhythmia database. *IEEE Engineering in Medicine and Biology*, 20(3):45–50, 2001.

[81] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[82] E. Mller, I. Assent, S. Gnnemann, R. Krieger, and T. Seidl. Relevant subspace clustering: Mining the most interesting non-redundant concepts in high dimensional data. In *IEEE International Conference on Data Mining*, pages 377–386, 2009.

[83] G. Nuel. Effective p-value computations using finite markov chain imbedding (FMCI): application to local score and to pattern statistics. *Algorithms for Molecular Biology*, 1(1):5–5, 2006.

[84] P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *IEEE International Conference on Data Mining*, pages 370–377, 2002.

[85] G. Pavesi, P. Mereghetti, G. Mauri, and G. Pesole. Weeder Web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. *Nucleic Acids Research*, 32(Web-Server-Issue):199–203, 2004.

[86] J. Pei, J. Han, B Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Mining sequential patterns by Pattern-Growth: The PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, 2004.

[87] P. A. Pevzner and S. Sze. Combinatorial approaches to finding subtle signals in DNA sequences. In *International Conference on Intelligent Systems for Molecular Biology*, pages 269–278, 2000.

[88] M. Régnier and A. Denise. Rare events and conditional events on random strings. *Discrete Mathematics and Theoretical Computer Science*, 6(2):191–214, 2004.

[89] John A. Rice. *Mathematical Statistics and Data Analysis*. Duxbury Press, Belmont, CA, 2nd edition, 1995.

[90] S. Robin and J. J. Daudin. Exact distribution of word occurrences in a random sequence of letters. *Applied Probability*, 36(1):179–193, 1999.

[91] S. Robin and S. Schbath. Numerical comparison of several approximations of the word count distribution in random sequences. *Computational Biology*, 8(4):349–359, 2001.

[92] S. Robin, S. Schbath, and V. Vandewalle. Statistical tests to compare motif count exceptionalities. *BMC Bioinformatics*, 8:84–93, 2007.

[93] D. Ron, Y. Singer, and N. Tishby. The power of Amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3):117–149, 1996.

[94] Matthias Schonlau. Masquerading user data. `http://schonlau.net/intrusion.html`.

[95] M. Z. Shafiq and A. X. Liu. A random walk approach to modeling the dynamics of the blogosphere. In *International IFIP TC 6 Conference on Networking*, pages 294–306, 2011.

[96] K. Shida. GibbsST: a gibbs sampling method for motif discovery with enhanced resistance to local optima. *BMC Bioinformatics*, 7:486–496, 2006.

[97] S. Sinha and M. Tompa. YMF: A program for discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucleic Acids Research*, 31(13):3586–35888, 2003.

[98] M. D. Smucker, J. Allan, and B. Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *ACM Internationl Conference on information and knowledge management*, pages 623–632, 2007.

[99] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38(1):1409–1438, 1958.

[100] D. Sourav and A. Bhattacharya. Mining statistically significant substrings using the Chi-Square statistic. *The Proceedings of the Very Large Database Endowment*, 5(10):1052–1063, 2012.

[101] J. D. Storey. A direct approach to false discovery rates. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(3):479–498, 2002.

[102] G. D. Stormo, T. D. Shneider, L. Gold, and A. Ehrenfeucht. Use of 'perceptron' algorithm to distinguish translational initiation sites in *E. coli*. *Nucleic Acids Research*, 10(9):2997–3010, 1982.

[103] P. Sun, S. Chawla, and B. Arunasalam. Mining for Outliers in Sequential Databases. In *SIAM International Conference on Data Mining*, pages 94–105, 2006.

[104] N. Tatti and J. Vreeken. The long and the short of it: summarising event sequences with serial episodes. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 462–470, 2012.

[105] G. Thijs, M. Lescot, K. Marchal, S. Rombauts, B. De Moor, P. Rouze, and Y. Moreau. A higher-order background model improves the detection of promoter regulatory elements by Gibbs sampling. *Bioinformatics*, 17(12):1113–1122, 2001.

[106] M. Thomas-Chollier, M. Defrance, A. Medina-Rivera, O. Sand, C. Herrmann, D. Thieffry, and J. van Helden. RSAT 2011: regulatory sequence analysis tools. *Nucleic acids research*, 39(Web Server issue):86–91, 2011.

[107] M. Tompa. An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. In *International Conference on Intelligent Systems for Molecular Biology*, pages 262–271, 1999.

[108] M. Tompa. Motif discovery for predicting regulatory elements. Department of Computer Science and Engineering, University of Washington, USA, 2011. Lecture Notes in CSE 427: Computational Biology.

[109] M. Tompa et al. Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, 23(1):137–44, 2005.

[110] J. van Helden, B. Andr, J. Collado-vides, Fijacin De Nitrgeno, and Universitg Libre De. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Molecular Biology*, 281(5):827–842, 1998.

[111] J. Wang and J. Han. BIDE: Efficient mining of frequent closed sequences. In *IEEE International Conference on Data Engineering*, pages 79–90, 2004.

[112] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[113] Geoffrey I. Webb. Discovering significant patterns. *Machine Learning*, 68(1):1–33, July 2007.

[114] L. Wei, E. Keogh, and X. Xi. SAXually explicit images: Finding unusual shapes. In *IEEE International Conference on Data Mining*, pages 711–720, 2006.

[115] L. Wei, N. Kumar, V. Lolla, E. J. Keogh, S. Lonardi, and C. Ratanamahatana. Assumption-free anomaly detection in time series. In *International Conference on Scientific and Statistical Database Management*, pages 237–240, 2005.

[116] I. H. Witten and T. C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, 1991.

[117] C. T. Workman and G. D. Stormo. ANN-Spec: a method for discovering transcription factor binding sites with improved specificity. *Pacific Symposium on Biocomputing*, 5:467–478, 2000.

[118] Y. Bin Y. Qiao, X. W. Xin and S. Ge. Anomaly intrusion detection method based on HMM. *Electronics Letters*, 38(13):663–664, 2002.

[119] K. Yamanishi and Y. Maruyama. Dynamic syslog mining for network failure monitoring. In *ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 499–508, 2005.

[120] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large databases. In *SIAM International Conference on Data Mining*, 2003.

[121] J. Yang, W. Wang, and P. S. Yu. Infominer: Mining surprising periodic patterns. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 395–400, 2001.

[122] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan. Detecting time series motifs under uniform scaling. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 844–853, 2007.

[123] M. J. Zaki. Sequence mining in categorical domains: Incorporating constraints. In *ACM International Conference on Information and Knowledge Management*, pages 422–429, 2000.

[124] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.

[125] J. Zhang, B. Jiang, M. Li, J. Tromp, X. Zhang, and M. Q. Zhang. Computing exact P-values for DNA motifs. *Bioinformatics*, 23(5):531–537, 2007.

[126] F. Zhu, X. Yan, J. Han, and P. S. Yu. Efficient discovery of frequent approximate sequential patterns. In *IEEE International Conference on Data Mining*, pages 751–756, 2007.

Appendices

Appendix A

List of single best motifs found by CPS on motif discovery benchmark

Datasets	**best motif (pattern|mismatch)**	**Correlation Coefficient**
yst01g	$ACGTGAACG\|2$	0.161317
yst02g	$GGCGCACTC\|2$	0.366508
yst03m	$CTGACTGCCT\|4$	0.143583
yst04r	$CGGGCTTCC\|2$	0.413408
yst05r	No motifs found	NaN
yst06g	$CCTAATTGG\|2$	0.459297
yst07m	No motifs found	1.000000
yst08r	$CCCAGACCG\|2$	0.317129
yst09g	$AGCCGCCGA\|2$	0.402748
yst10g	No motifs found	1.00000
hm01g	$ATCGAT\|1$	0.0003
hm02r	$ACACACACA\|2$	-0.022845
hm03r	$ATCTATCTA\|2$	-0.004013
hm04m	$TCGATCGCC\|2$	-0.008672
hm05r	$GCGCCGTCG\|2$	-0.042911
hm06g	$CACGTGACC\|2$	0.428752
hm07m	$TAGCCATGG\|2$	0.111794
hm08m	$CTAACGTCA\|2$	0.464688
hm09g	$ATCTATAAA\|2$	-0.012551
hm10m	$CTTTTCGCG\|2$	0.359909
hm11g	$CGTCGT\|1$	0.020063
hm12r	$TATAGGACT\|2$	0.242943
hm13r	$ATCATTAAC\|2$	0.309013
hm14	$ATCTATTTA\|2$	0.199734
hm15r	$AAATCGGAA\|2$	-0.015190
hm16r	$TACGTA\|1$	0.019723
hm17g	$TCGGGAATT\|2$	0.453401
To be continued		

Table A.1 – *Continued*

Datasets	**best motif (pattern|mismatch)**	**Correlation Coefficient**
hm18m	$CGCGAATCG\|2$	-0.008618
hm19g	$AGCGACGGG\|2$	-0.032698
hm20r	$ATCGAACTC\|2$	-0.01
hm21g	$CCGGTCGCT\|2$	-0.020455
hm22m	$CGCACGTGA\|2$	0.623910
hm23r	$GGCAGGATGG\|3$	0.121116
hm24m	$CGTTTCGGG\|2$	0.134159
hm25g	$CTATAGTAC\|2$	0.304641
hm26m	$CCGTGACGC\|2$	0.025258
mus01r	No motifs found	NaN
mus02r	$TAAGACTAA\|2$	-0.021814
mus03g	$CCGCCACTCC\|4$	0.014435
mus04m	$TTGCGCAA\|2$	0.211805
mus05r	$GTAAAACCA\|2$	0.088244
mus06g	$ACACAAGGG\|2$	0.108647
mus07g	$CGNNNN\|1$	-0.004119
mus08m	No motifs found	NaN
mus09r	$TATGTAAAA\|2$	0.587622
mus10g	$ACTCCGCAC\|2$	0.214365
mus11m	$GCGCGGGCG\|2$	0.385669
mus12m	$TTTCCAATG\|2$	0.179836
dm01g	$GGGATCGCT\|2$	0.043744
dm02g	$CTTAACCCG\|2$	0.138071
dm03m	$TGGTTAACG\|2$	-0.018730
dm04g	$CGCACGTAC\|2$	-0.000856
dm05g	$GTCCGCGTT\|2$	-0.020431
dm06r	$TAGATACCT\|2$	-0.032318
dm07m	$GTGAGGGTT\|2$	NaN
dm08m	$TACCGTTAG\|2$	NaN

Appendix B

Conjecture: Constructing a Core Pattern Set is NP-Hard

As it was formalized in Definition 10, the Core Pattern Set for a set of significant patterns O is a subset E that *explains* (Definition 7) all significant patterns in O, and that has the minimum cardinality among all such subsets. In this section, we show that the decision version of the Core Pattern Set problem is likely to be *NP-complete*. The main intuition behind this conjecture is that the decision version of the Core Pattern Set problem is very similar to the problem of the Dominating Set problem.

In graph theory, a *dominating set* for a graph $G = (V, E)$ is a subset D of V such that every vertex not in D is adjacent to at least one member of D. The *domination number* $\gamma(G)$ is the number of vertices in a smallest dominating set for G. The decision version of the dominating set problem concerns testing whether $\gamma(G) \leq K$ for a given graph G and input K; it is a classical NP-complete decision problem in computational complexity theory [42]. Therefore it is believed that there is no efficient algorithm that finds a smallest dominating set for a given graph. In the following, we first prove that the dominating set problem is also NP-Complete in a directed graph. Then, we present our conjecture on the NP-Completeness of the core pattern set problem.

Theorem 15 *Given a directed graph $G = (V, E)$ and an integer K, the problem of deciding whether G contains a dominating set of size at most K is NP-Complete.*

Proof. *We give a proof of NP-completeness by reducing the set cover problem to the dominating set problem. It is easy to prove that the directed dominating set is in NP. For a given solution, we just need to verify that each node of the graph is either in the solution, or is connected by a node in the solution through an incoming edge. This can be done in polynomial time.*

Reduction*: The reduction from the set cover to the dominating set problem in the directed version works similar to the case of an indirected graph. Let (S, U) be an instance of the set cover problem with the universe U and the family of subsets $S = \{S_i : i \in I\}$. Construct a directed graph $G = (V, E)$ as follows:*

- *The set of vertices is $V = I \cup U$*
- *There is a directed edge $(i, j) \in E$ between each pair $i, j \in I$. There is also a*

directed edge (i, u) for each $i \in I$ and $u \in S_i$. This results in I being a clique and U being an independent set.

Now, if $C = \{S_i : i \in D\}$ is a solution of the set cover problem for some subset $D \subseteq I$, then D is a dominating set for directed graph G, with $|D| = |C|$. The reason is that for each node $u \in V$ two cases are possible. If $u \in U$, there is an index $i \in D$ such that $u \in S_i$, and due to the reduction properties, there is an edge between i towards u in the directed graph G. In the case that $u \in I$, every such a node is either in D or has an incoming edge from a node in D, again due to the fact that I is a clique, and $D \subseteq I$.

Conversely, let D be a dominating set for the directed graph G. Then, we can construct another dominating set X such that $|X| \leq |D|$, and $X \subseteq I$. In other words, we can construct a dominating set of size at most $|D|$ such that every node in this dominating set are selected from the set of nodes in I. To do so, we simply replace each $u \in D \cap U$ with a node in $i \in I$ so that there is a directed edge from i to u (there should be at least one node in I with this property or no set cover is feasible for this instance of the problem). Then, $C = \{S_i : i \in X\}$ is a feasible solution of the set cover problem, with $|C| = |X| \leq |D|$. ■

An example reduction from the set cover problem to the dominating set problem is illustrated in Figure B.1. In this example, the union set is defined as $U = \{a, b, c, d, e\}$, and $I = \{1, 2, 3, 4\}$. Four different sets are defined as $S_1 = \{a, b, c\}$,$S_2 = \{a, b\}$, and $S_3 = \{b, c, d\}$, and $S_4 = \{c, d, e\}$. Based on these definitions, the set $C = \{S1, S4\}$ is a set cover. This corresponds to the dominating set $D = \{1, 4\}$.

The decision version of the core pattern set problem is very similar to the minimum dominating set problem, as we have to decide if all the patterns in a set of significant patterns can be explained by a subset of cardinality at most K. This will lead to the following conjecture, whose proof is left as an open problem.

Conjecture 16 *For a set of significant patterns O, the problem of deciding whether a subset E of size at most K exists which explains (as formalized in Definition 7) all patterns in O is NP-Complete.*

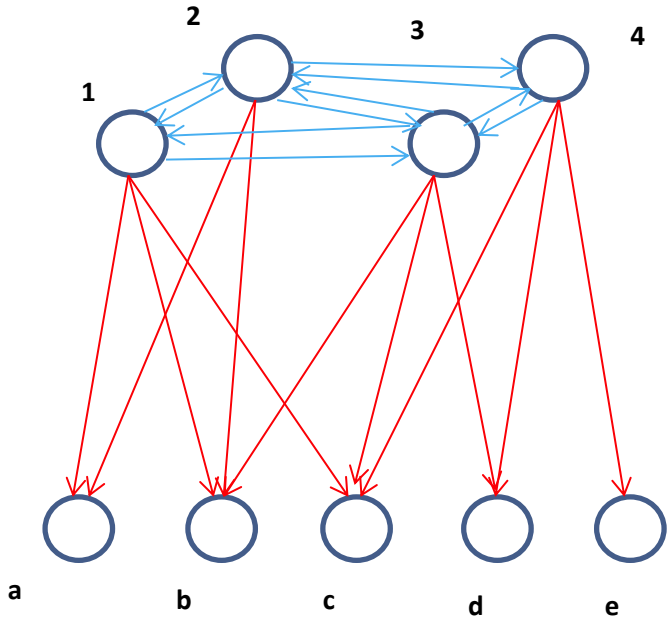


Figure B.1: Set covert problem to dominating set problem reduction: An illustration

Appendix C

Speeding-up Strategies and Implementation Issues

In this section, we present some of the techniques employed for speeding-up the running time of our algorithm, by speeding-up the P-value calculation and the core pattern set construction steps of our algorithm.

C.1 Early Abandonment of Exact P-value Computation

We use Formula 4.2 for calculating the exact p-values and finding statistically significant patterns in the first step of our algorithm. The values of functions $f(i, j)$ are stored in a two-dimensional matrix with l rows and t columns, where l is length of the data sequence, and t is the observed frequency of the pattern in the data sequence. The cells of the matrix can be filled horizontally (smallest rows first) or vertically (smallest columns first) following a dynamic programming technique. As shown in the formula 4.2, the p-value of a word W is equal to the sum of values in the last row (*i.e.* the row with index t). In calculating the p-values, when evaluating the significance of a pattern or in the case of testing for an explain relation where the p-value of a pattern should be computed in the presence of a constraint set, we are only interested to determined if the p-value is smaller than a significance level α. Because we are only interested in patterns whose p-values are less than a significance level α (usually a very small value such as 0.01), we can fill the matrix vertically and abandon the calculations as soon as the sum of the cells in the last row exceeds the significance level α. This strategy could save unnecessary calculations especially in a long data sequence. The same strategy can be used in formula 4.5 for calculating p-values in a sequence with a constraint set.

C.2 Approximating P-values

Calculating the exact p-values of patterns could be a time consuming task when the input sequence is very long and/or the size of the alphabet (*i.e.* number of unique symbols) is very large. Unfortunately, exact results are not adapted in practice for long sequences because of heavy numerical calculation, but they allow the user to assess the quality of the stochastic approximations when no approximation error

can be provided.

Even though the early abandoning strategy improves the running time for some patterns, an exact p-value calculation seems impractical if we have to do it for a very large number of candidate *surprising patterns*. In practice, approximation techniques are used to compute the p-value of patterns in a reasonable time [92, 98]. Three common approximation approaches to computing p-values which are used in practice include the *Gaussian*, the *Poisson*, and the *Compound Poisson* approximation.

We are concerned with the distribution of counts of a k-letter word $W = w_1...w_k$ in a sequence S of length ℓ. Let N be a random variable representing the number of counts of a k-letter word W. For a given word W with observed frequency t, we are interested in the probability distribution function $pdf(N,t) = Prob(N = x), \forall x \geq 0$, and the cumulative distribution function $C(N,t) = P(N \leq t)$.

Let $\pi(W)$ represents the probability of occurrence of W at any location of sequence S. At it was presented in Section 4.1 and equation 4.2, the exact distribution of counts of W can be obtained via the distribution of the position i of the n-th occurrence of W in S. The computation time is in $O(ktlog(\ell))$ in most cases, but for the sake of numerical stability, an elementary algorithm of complexity $O(kt\ell)$ is often more satisfying [91], where k is the length of the work W, ℓ is the length of the sequence S, and t is the observed frequency of the word W in the sequence S.

The distribution of counts of the word W, represented by the random variable N, depend on the *overlapping* structure of the word W, which are captured in some approximation techniques such as the *Gaussian* approximation. The period of a word W is defined as follows:

Definition 17 *A period of the word $W = w_1w_2...w_k$ is an integer $p \in \{1, ..., k-1\}$ such that $w_i = w_{i+p}, \forall i = 1, ..., k-p$. We denote the set of periods of W by $\mathcal{P}(W)$.*

Let $\mathcal{P}'(W)$ be the set of the principal periods of W, *i.e.* the periods that are not strictly multiple of a minimal period of W. We define the quantities $A_i(W)$ and A_W as follows:

$$\begin{aligned} A_i(W) &= \pi(w_1...w_i) \qquad \text{for} \quad i = 1, ..., k \qquad \text{(C.1)} \\ A(W) &= \sum_{p \in \mathcal{P}'(W)} A_p(W) \end{aligned}$$

C.2.1 Gaussian Approximation

Kleffe *et al.* show that the *first* (mean) and the *second* (variance) *moments* of counts of W can be computed using the following formulas [61]:

$$\mathbb{E}_\mu(N) = (\ell - k + 1)\pi(W) \qquad \text{(C.2)}$$

$$\begin{aligned} \mathbb{V}_\mu(N) &= \mathbb{E}_\mu(N) + 2\sum_{p \in \mathcal{P}}(\ell - k - p + 1)\pi(w_1...w_pw_1...w_k) \\ &\quad + \pi^2(W)[\ell(1-2k) + 3K^2 - 4k + 1] \qquad \text{(C.3)} \end{aligned}$$

where $\mathcal{P}'(W)$ is the set of the principal periods of W. Using the *central limit theorem* [89], the distribution of N converges to a *Gaussian* distribution as the length of the sequence grows to ∞. The distribution of N is then approximated by the *Gaussian* distribution with mean $\mathbb{E}_\mu(N)$ and variance $\mathbb{V}_\mu(N)$, as follows:

$$P(N = t) = \frac{1}{\sqrt{2\pi \mathbb{V}_\mu(N)}} e^{-\frac{(t-\mathbb{E}_\mu(N))^2}{2\mathbb{V}_\mu(N)}} \tag{C.4}$$

which gives us a simple tool to compute the p-value of a word W with an observed frequency of t using the cumulative probability function. The *Gaussian* approximation of the count of a word is based on the fact that the expected count is a linear function of the length of the sequence, which is valid only for frequent words.

C.2.2 Poisson Approximation

The count N of a word occurrence can be approximated using a *Poisson* distribution, where the *events* (*i.e.* word occurrences) are assumed to be independent of each other, and the random variable N represents the number of events among a total number of trials (i.e. length of the data sequence). The approximation comes from the fact that the probability of occurrence of words are independent from each other, where in practice it does not hold.

Based on this model, the p-values can be approximated using the *Poission* distribution ($N \sim \rho(\lambda)$), with λ as the parameter of the distribution being equal to $\ell \times \pi(W)$, where ℓ is the length of the input sequence and $\pi(W)$ is the probability of occurrence of the word W at any location.

The p-value of a word W with an observed frequency t in the data sequence can be approximated using the following formula [16]:

$$P(N \geq t) = P(\rho(\lambda) \geq t) = 1 - e^{-\lambda} \sum_{i=0}^{t-1} \frac{\lambda^i}{i!} \tag{C.5}$$

Among the presented approximation methods, we implemented the *Gaussian* and the *Poisson* approximation techniques for computing p-values. We adopted to use the *Poisson* approximation because it proved to be much faster than the *Gaussian* approximation in our experiments. Approximations of p-values can be used in a filtering step for exact p-value calculations. In our method, the maximum difference ϵ between an exact p-value and its approximate value based on the *Poisson* distribution is estimated using a subset of the data. Then, a two-step filter-refinement procedure is used for selecting statistically significant patterns. First, the p-value of a pattern is approximated using the *Poisson* distribution. If the approximate p-value is less than $\alpha + \epsilon$, the exact p-value of the pattern is calculated for a statistical test w.r.t the significance level α. Otherwise, the pattern will be considered as not being statistically significant.

Given the fact that in real-world settings most of the subsequences in the data are normal, an approximation using a method such as *Poission* distribution will avoid costly exact p-value calculations for a large number of patterns in the data. It is worth mentioning that in our experimental evaluation (Section 7), we use approximate p-values as filter step only for experiments on the Masquerading user dataset, in which the large number of symbols in the dataset made an exact p-value

calculation too time consuming. In other experiments, we calculated the exact p-values using the early abandoning strategy. Our approach to calculating the p-values in the extended model (i.e. *mismatch string model*), which is also based on a Poission distribution, was discussed in more details in Chapter 5.

C.3 Exploiting Pattern Locality in the Explain Relation

In the pseudocode provided in Section 4.2, every pair of patterns p_i and p_j in the set of unexplained patterns P_{rest} are considered, to see whether adding a pattern such as p_i to the current solution can explain the significance of another pattern p_j, *i.e.* we check if $(P_{sol} \cup \{p_i\}) \succ p_j$. This involves p-value calculations with a new constraint set for each pair in P_{rest}, which is a time consuming task.

The information about overlapping between patterns can be exploited to avoid unnecessary p-value calculations. The idea is that if two patterns p_i and p_j don't have any *overlap* (*i.e.* no prefix/suffix of p_i is a suffix/prefix of p_j and none of the patterns is a substring of the other one), and also the current solution cannot explain the significance of the pattern p_j, then the new solution which is constructed by adding the pattern p_i to the existing solution (i.e. $(P_{sol} \cup \{p_i\})$) cannot explain the pattern p_j as well. This justification is formalized in the following theorem:

Theorem 18 *Assume that a set O of patterns cannot explain the significance of another pattern p_j. If a pattern $p_i \notin O$ does not intersect the pattern p_j, then the set of patterns $N = O \cup \{p_i\}$ cannot explain the significance of p_j.*

Proof. *In order to prove that the set N cannot explain the significance of the pattern p_j, we have to show that $Prob(frequency(p_j) \geq t|C_{N,S}) \leq \alpha$, where S is the data sequence, t is the observed frequency of p_j in S, $C_{N,S}$ is the constraint set resulting from the set N, and α is the significance level used for statistical tests.*

We use the Poisson Binomial distribution, as introduced in Chapter 5, to approximate the p-value of the pattern p_j in the presence of a constraint set. For simplicity, we assume that the input sequence has a uniform background distribution.

Let $Q = \{q_1, q_2, ..., q_n\}$ and $R = \{r_1, r_2, ..., r_n\}$ represent the success probabilities of occurrence of the pattern p_j in locations $l = 1, ..., n$ of sequence S under constraint sets $C_{O,S}$ and $C_{N,S}$, respectively, where n is the length of the input sequence S. Using the fact that the patterns p_i and p_j don't have any intersection, it is easy to verify that $q_l \geq r_l$, where $1 \leq l \leq n$. The reason is that the constraint set $C_{N,S}$ includes all the constraints specified by $C_{O,S}$, as well as new constraints that are due to addition of the new pattern p_i to the constraint set $C_{O,S}$. Let I represent the set of indices occupied by instances of the pattern p_i in sequence S. The fact that the constraint set $C_{N,S}$ should comply with symbols in S at all locations specified by I, and the fact that the patterns p_i and p_j cannot have any intersection with each other, imply that the probability of occurrence of p_j is equal to 0 at all indices specified by I under the constraint set $C_{N,S}$. On the other hand, the probability of occurrence of p_j is equal to $Prob(p_j) > 0$ at all indices specified by I under the constraint set $C_{O,S}$. Therefore,

$$Prob(frequency(p_j) \geq t|C_{N,S}) \leq Prob(frequency(p_j) \geq t|C_{O,S}) \qquad \text{(C.6)}$$

On the other hand, the fact that the set O of patterns cannot explain the significance of the pattern p_j implies that

$$Prob(frequency(p_j) \geq t | C_{O,S}) \leq \alpha \tag{C.7}$$

Using C.6 and C.7, we can conclude that $Prob(frequency(p_j) \geq t | C_{N,S}) \leq \alpha$. This completes the proof of the theorem.

■

The useful consequence of the this theorem is that it can help reduce the scope of explain relation tests for a pattern. In other words, in the process of testing the patterns that can be explained by a new set of patterns, the search space can be limited to the set of patterns which have intersection with the newly added pattern (*i.e.* p_i) to the new solution.

Example 19 *In the Example 11, and for* 3 *patterns* $A =$ "44334334", $B =$ "55535", *and* $C =$ "5535", *the explain relation tests* $(\{A\} \succ \{B\})$? *and* $(\{A\} \succ \{C\})$? *can be skipped based on the theorem 18 and the fact that the pattern A does not have any intersection with patterns B and C.*

It is worth mentioning that the intersection property based on what formalized in theorem 18 might not be as helpful as the general model (*i.e.* exact string matching) in avoiding the *explain* relation tests in the extended model (*i.e.* mismatch string model). The reason is that the proposed *mismatch model* for motifs allows a mismatch at any location of a pattern as long as the total number of mismatches is less than a threshold. Therefore, in order to exclude an *explain relation* test $U \succ \{P\}$, we have to make sure that the pattern P does not have an intersection with any of the patterns in the set U considering all possibilities of mismatches in the patterns contained by set U.

C.4 Using Prefix Tree to Speed-up Probability Computations

The statistical significance of patterns depends on the probability of occurrence of the patterns at any location of the sequence, given the underlying model that is learned using the training data. No matter which measure is used for evaluating the statistical analysis of the patterns (*e.g.* *p-value*, *E-value*, *information content*), we we need to compute the probability of occurrence for a large number of potential significant patterns. This involves an extensive computation for long sequences. Efficient computations of these probabilities is crucial in the overall running time of a method.

We use a *Prefix tree* data structure to speed-up the computation process. The application of a prefix tree data structure for p-value computation speed-up is based on two main observations: 1) many patterns share the same prefix; and 2) calculation of the probability of a pattern based on a Markov chain model will depend on the probability of their prefixes. This makes a *prefix tree* a suitable data structure for reducing the time complexity and space complexity.

A *Tri*, or a *Prefix Tree*, is an ordered tree data structure that is used to store a dynamic set of keys which are usually strings, and have the potential of sharing a prefix. The root of the prefix tree is associated with the *empty* string. All the descendants of a node have a common prefix of the string associated with that node, and each node has a maximum fan-out equal to the length of the alphabet. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated. It can be verified that the widely-used operations 'insert' and 'lookup' can be implemented in $O(m)$, where m is the length of the pattern.

We use the nodes of the prefix tree to store additional data, in addition to the *key* of the node which corresponds to a symbol in the alphabet $\sum$. In each node V of the tree, we store the probability of occurrence of the pattern corresponding to the list of symbols from the root to node V. Figure C.1 shows an example of a prefix tree used for storing the probability of occurrence of patterns.

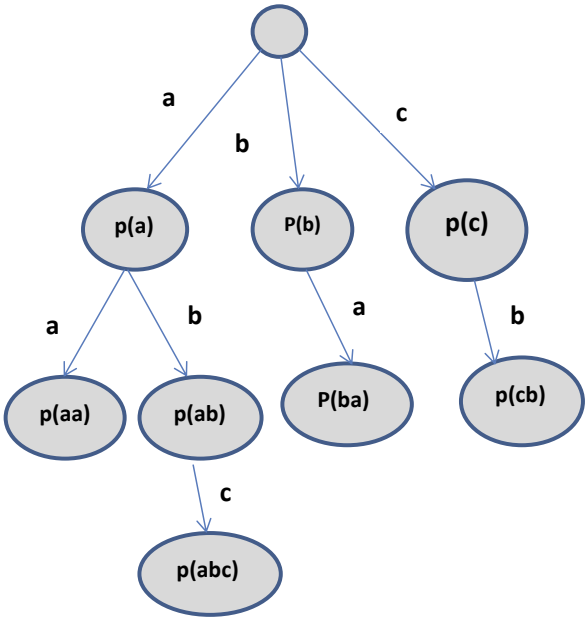


Figure C.1: A prefix tree data structure storing the probability of patterns.

For a new pattern $W = w_1 w_2 ... w_m$, we first query the prefix tree data structure to see if the pattern is already in the tree. The *lookup* involves matching the symbols of the pattern W against the branches of the tree until either all the symbols of the pattern W have been consumed, in which we have a *hit*, or until the point where no child node can be matched against the current symbol in the pattern W, in which we have a *mismatch*. If the pattern is already stored in the tree, we are done. Otherwise, we insert the pattern in the tree from the last node in the tree which has been matched against the current symbol in the pattern.

Example 20 *Assume that we are going to insert a new pattern $W =$ "abcda" in the prefix tree example in Figure C.1. Also, assume that the underlying distribution of*

the data is represented by a Markov chain model of order 2*. The insertion procedure of pattern* W *in the tree consists of the following steps*

1. *The substring* $Q =$ *"abc" matches with a path in the tree, ending in node* V*, and as a result no branching is required up to symbol* c*.*

2. *No child node of* V *matches the next symbol of* W*, which is* d*. As a result, a new node* U *is created under node* V *with connecting edge* d*. The probability of occurrence of substring* $X =$ *"abcd" is equal to* $P(Q) \times P(d|bc)$*, assuming a Markov chain of order* 2*, which is easily derived from the probability transition matrix of the Markov chain model. The computed probability for substring* X *is stored in the newly created node* U *(as shown in Figure C.2).*

3. *Again, no child node of* U *matches the next symbol of* W*, which is* a*. As a result, a new node* Z *is created under the node* U *with connecting edge* a*. The probability of occurrence of the substring "abcda" is equal to* $P(X) \times P(a|cd)$*. The computed probability is stored in the newly created node* Z *(as shown in Figure C.2). This completes the insertions procedure. The new prefix tree after the insertion is shown in Figure C.2.*

C.5 Parallel Implementation on the Westgrid Cluster

Computing the probability of occurrence for patterns with larger mismatch degrees is a time-consuming task, despite all the improvements achieved through the techniques proposed in Section C.4. To remedy this problem, and to improve the running time of our proposed method with the extended mismatch model, we investigated the parallel implementation of our proposed framework. The parallel implementation was used to run our experiments on the motif discovery benchmark, which will be discussed in Section 7.5.

Due to the fact that the probability computations of mismatch patterns are the bottleneck of our method, we tried to identify the components which rely on theses probability computations, and make them as parallel as possible. Referring to the Figure C.3 which represents the components of our proposed framework, we can see that the *Statistical Test Analysis* and *Core Pattern Set Construction* components involve computing p-values of patterns, which in turn relies on computing the probability of occurrence of a pattern. These two components are redesigned for parallel implementation.

C.5.1 Parallel Statistical Test Analysis

The goal of this component is to compute the p-values of all extracted patterns from the sequence data. In the parallel implementation, we divided this component into two sub-components which run sequentially, as follows:

- **Fast, Selective P-value Computation**: In this component, we scan the extracted patterns, and compute the p-value of the patterns for which the computation can be completed fast (*i.e.* less than a few seconds). The computation speed is mainly derived by the number of mismatches and the length

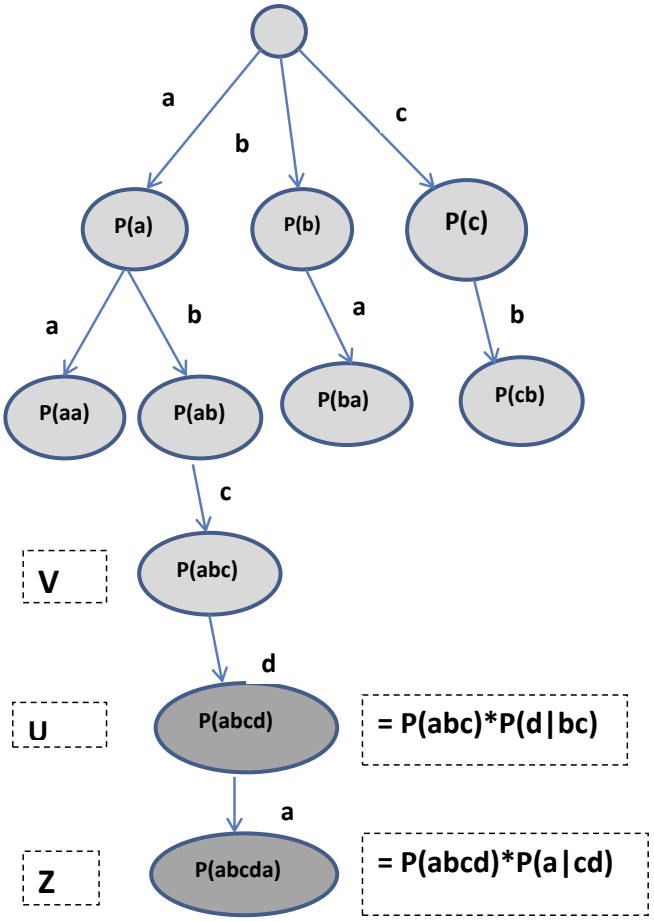


Figure C.2: The prefix tree data structure C.1 after adding the pattern $W =$"abcda".

of the pattern. In our experiments on motif discovery benchmark, we calculated the p-value for patterns whose length is less than 10 and the maximum number of mismatches are less than 3. The remaining patterns are passed to the next step to be processed in parallel.

- **Costly P-value Calculators**: A separate job is created for patterns whose p-value computation is time-consuming, and is assigned to a *costly p-value calculator* component. These components can be executed in parallel, and contribute significantly to reducing the running time for heavy p-value computations. In Figure C.3, these components are represented by small computation boxes chained in a row in the Statistical Analysis Test component.

C.5.2 Parallel Core Pattern Set Construction

The *Core pattern set construction* is another component of our framework which relies heavily on p-value computations. In each step of our greedy search strategy, we want to select the best *explaining pattern* (*i.e.* q^*), which explains the largest number of remaining patterns. In each step of the greedy search algorithm, the number of patterns explained by a specific pattern A, in the presence of a constraint set, is independent of the number of patterns which can be explained by another pattern B (Conceptually, we might be able to draw some conclusions on one explain count from another explain count, but the way the explain relations have been implemented in our framework, they are independent of each other). Therefore, these processes can be executed in parallel.

In the parallel implementation of the *core pattern set* component, two sub-components are identified as follows:

- **Explain Relation Testers**: In each step of the greedy search algorithm, a separate job is created to count the number of patterns explained by any pattern in the set P_{rest}. Every such a job is assigned to a sub-component *explain relation tester*, which is executed in parallel to other sub-components of this type. In Figure C.3, these components are represented by computation boxes chained in a row in the Core Pattern Set Construction component.

- **Aggregator**: The goal of an *aggregator* component is to select the *best explaining* pattern q^*, which explains the largest number of remaining patterns, based on the output of the *explain relation tester* jobs for different patterns. Upon selection of the *best* pattern q^*, the sets P_{sol} and P_{rest} are updated, and passed through the next step of the greedy search algorithm. In Figure C.3, these components are represented by wide boxes in the Core Pattern Set Construction component, which get their input from all the explain relation testers in the same step of the search algorithm.

The parallel implementation of the *core pattern set* component involves alternate execution of sub-components *explain relation testers*, which are run in parallel, and the *aggregator*. The algorithm stops when the set P_{rest} becomes empty after an execution of an *aggregator* sub-component.

The parallel architecture of our *core pattern set* framework is represented in Figure C.3. The parallel components of *heavy p-value calculators* and *explain relation testers* are denoted in a single horizontal layer, and are connected to each other in dotted lines.

We used the Westgird Canada computational resources for running our parallel *core pattern set* framework. The Westgrid framework provides utilities for submitting, scheduling, and allocating the resources for jobs. In particular, we used the "grex" and "hermes" clusters for running our experiments on the Westgird Canada.

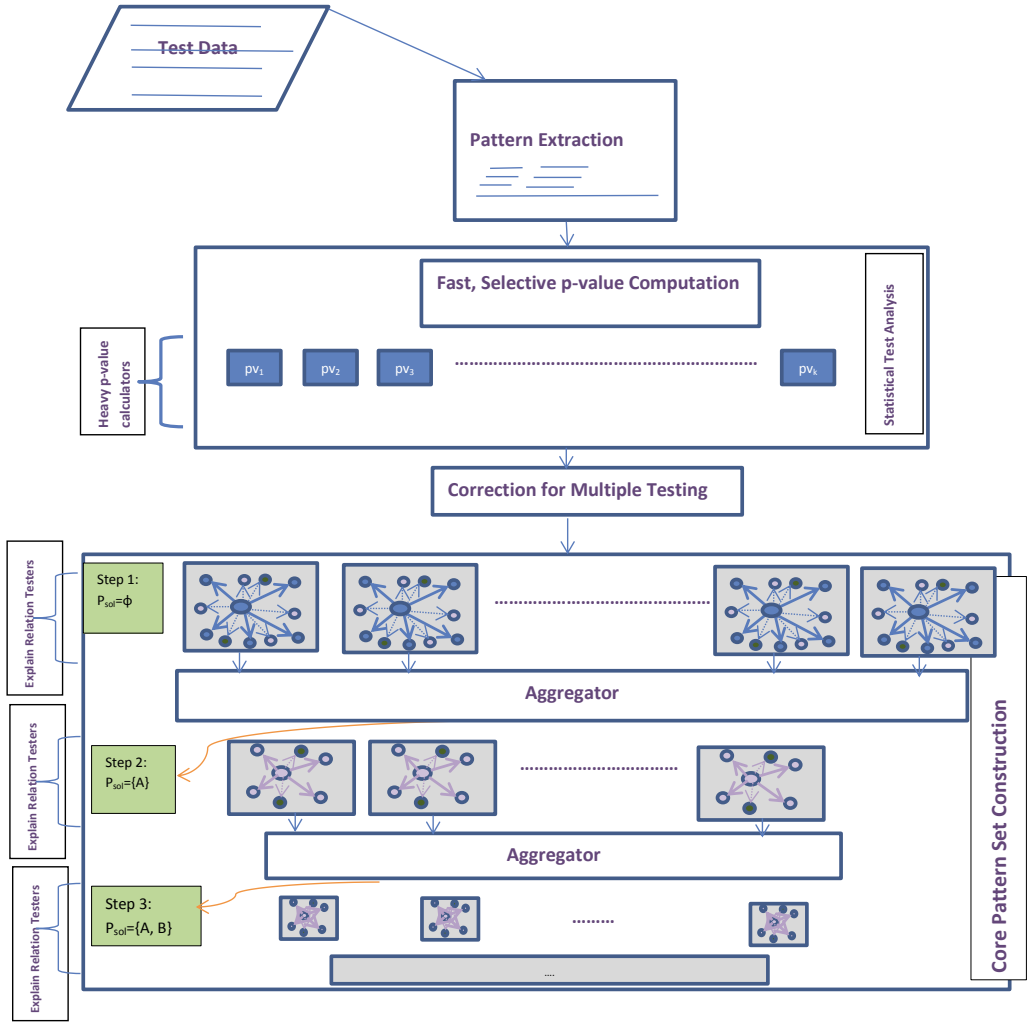


Figure C.3: *Parallel Implementation of Our Framework on Westgrid Canada.*