Combinatorial and Probabilistic Approaches to Motif Recognition

by

Christina Anne Boucher

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2010

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Short substrings of genomic data that are responsible for biological processes, such as gene expression, are referred to as motifs. Motifs with the same function may not entirely match, due to mutation events at a few of the motif positions. Allowing for non-exact occurrences significantly complicates their discovery. Given a number of DNA strings, the motif recognition problem is the task of detecting motif instances in every given sequence without knowledge of the position of the instances or the pattern shared by these substrings.

We describe a novel approach to motif recognition, and provide theoretical and experimental results that demonstrate its efficiency and accuracy. Our algorithm, MCL-WMR, builds an edge-weighted graph model of the given motif recognition problem and uses a graph clustering algorithm to quickly determine important subgraphs that need to be searched further for valid motifs. By considering a weighted graph model, we narrow the search dramatically to smaller problems that can be solved with significantly less computation.

The Closest String problem is a subproblem of motif recognition, and it is NP-hard. We give a linear-time algorithm for a restricted version of the Closest String problem, and an efficient polynomial-time heuristic that solves the general problem with high probability. We initiate the study of the smoothed complexity of the Closest String problem, which in turn explains our empirical results that demonstrate the great capability of our probabilistic heuristic. Important to this analysis is the introduction of a perturbation model of the Closest String instances within which we provide a probabilistic analysis of our algorithm. The smoothed analysis suggests reasons why a well-known fixed parameter tractable algorithm solves Closest String instances extremely efficiently in practice.

Although the Closest String model is robust to the oversampling of strings in the input, it is severely affected by the existence of outliers. We propose a refined model, the Closest String with Outliers problem, to overcome this limitation. A systematic parameterized complexity analysis accompanies the introduction of this problem, providing a surprising insight into the sensitivity of this problem to slightly different parameterizations.

Through the application of probabilistic and combinatorial insights into the Closest String problem, we develop sMCL-WMR, a program that is much faster than its predecessor MCL-WMR. We apply and adapt sMCL-WMR and MCL-WMR to analyze the promoter regions of the canola seed-coat. Our results identify important regions of the canola genome that are responsible for specific biological activities. This knowledge may be used in the long-term aim of developing crop varieties with specific biological characteristics, such as being disease-resistant.

Dedication

This thesis would be incomplete without a mention of the support given by the members of the Waterloo Potters' Workshop. Their encouragement and creativity has inspired me to share, explore and implement my ideas, whether it be on paper or with clay.

Acknowledgements

This research project would not have been possible without the support of many people. I am extremely grateful to my supervisors, Ming Li and Prabhakar Ragde, who were abundantly helpful and offered invaluable assistance, support and guidance. Ming Li gave me with outstanding support for my research. He provided me with great insight into my research topic, as well as fruitful collaborations; both helped broaden my view of the research field and led me to exciting research directions. Prabhakar Ragde provided invaluable feedback that greatly helped improve my thesis. His criticism taught me how to present my work in a clear, more concise manner.

My deepest gratitude are also due to the members of the supervisory committee: Ming Li, Prabhakar Ragde, Anne Condon, Bin Ma, and Jonathan Buss. Bin Ma's knowledge and insights in this research study were invaluable. Discussions with him led to inspired exciting new research. Anne Condon provided me with encouragement and support, and is an academic role-model for me. Professors Charles Clarke, Naomi Nishimura, Nick Wormald, Tamer Özsu, and Yuying Li are also dearly mentioned.

I am thankful to all my coauthors and collaborators. I had the great opportunity to work with research members of Microsoft Research in Cambridge and the University of Manchester; notably, Christopher Bishop, John Winn, Markus Svensén, Angela Smith, and Adnan Custovic. I am grateful to my collaborators of the Alberta Research Council, Limin Wu and Saleh Shah, who provided a biological application of my algorithmic work. I would like to acknowledge the administrative staff, especially Margaret Towell and Wendy Rush.

I would also like to convey thanks to the Natural Sciences and Engineering Research Council (NSERC) of Canada, the Anita Borg Institute for Women and Technology, Google, and David R. Cheriton for providing the financial means and laboratory facilities. This research was supported by the NSERC grants of Prabhakar Ragde, Dan Brown and Ming Li.

I would like to thank all my friends for making my stay in Waterloo an enjoyable one. I will fondly remember coffee breaks, pints of beer at Kick Offs and the Graduate House, GSA stuff, missing epic Halloween parties, knitting sessions, and the parties. Thanks to Kathleen Wilkie, Greg Zaverucha, Rob Warren, Cora Borradaile, Andrea Bunt, Steph Durocher, Jeff Dicker, Patrick Kling, Andrew Brown, Irene Pivotto, Peter Nelson, Robin Christian, Michael LaCroix, David Pritchard, Craig Sloss, Alex Hudek, Babak Alipanahi and Paul Church. Montreal friends: Emily Austin, Christine Caruso and April Rose.

I would like to thank my dear parents. My father was the first to introduce me to University of Waterloo and the field of computer science. Lastly, I'd like to thank my best friend and husband, Jaime Ruiz, who has been a relentless source of support for me.

He very generously took the time to listen to my presentations, proofread my papers and scholarship applications, and take care of household duties.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In 1977 Sanger *et al.* [97] published their results describing the usage of dideoxynucleotide termination DNA sequencing technology and, thus, formed formed the basis for DNA sequencing. Since this initial development there have been countless improvements in sequencing technology by drawing on advancements in molecular biology, chemistry, and computer science. With the development of high-throughput next generation sequencing technologies has arisen large amounts of genomic data, and an increased need for novel ways to analyze this data. The growing amount of data makes it impossible to analyze DNA sequences manually.

This revolution in DNA sequencing technologies has led to many new challenging problems in bioinformatics. For example, the human genome contains over three billion building blocks called nucleotides, which are represented by the symbols A, C, G, and T. Due to the immense size of this dataset, it is infeasible to analyze this data manually and biologists must use computational methods to find novel, functional regions. In order to begin to answer some of these problems in bioinformatics, we need to abstractly and realistically model these problems as discrete objects. This thesis will focus on one such model, the development of efficient algorithms for that model, and the application of these algorithms to analysis of genomic data.

Short substrings of genomic data that are responsible for biological processes, such as gene expression, are referred to as *motifs*. Motifs with the same function may not entirely match, due to mutation events at a few of the motif positions. Given a number of DNA strings, the motif-recognition problem is the task of discovering motif instances in every given string without knowledge of the position of the instances or the pattern shared by these substrings. Motif recognition is a NP-complete problem, and therefore it is unlikely that there exists a polynomial-time algorithm for this problem. Since finding motif instances using laboratory methods is an extremely costly and lengthy process, it is vital to create accurate and efficient computational methods to assist in finding possible motif

instances. The aim is to develop an application that will find possible motif instances; the user then must decide whether these instances warrant further biological investigation based on their statistical significance. Numerous algorithms and programs have been developed to find motifs in DNA sequences but many have limited accuracy or computational tractability [24, 88].

In Chapter 3, we describe a novel approach to motif recognition, and provide theoretical and experimental results that demonstrate its efficiency and accuracy. Our algorithm, MCL-WMR, builds a weighted graph model of the given motif-recognition problem and uses a graph clustering algorithm to quickly determine important subgraphs that need to be searched further for valid motifs [16]. Previous algorithms and programs search exhaustively or probabilistically on an unweighted graph or sequence model of the input data, but due to the lack of information contained in these models, the required search is extremely broad and requires considerable computation time. By considering a weighted graph model, we narrow the search dramatically to smaller problems that can be solved with significantly less computation. An empirical comparison to previous work illustrates that MCL-WMR is a leading method with respect to its running time and accuracy. This chapter is based on joint work with Daniel G. Brown and Paul Church [16].

The Closest String problem is a NP-complete problem related to the motif-recognition problem. In Chapter 4 we study combinatorial and probabilistic algorithms for this problem, and consider two subgroups of the problem instances: where the number of strings is small, and where the number of strings is relatively large. We give a linear-time algorithm for the restricted version of Closest String where the number of strings is small, a result which resolves an open problem described by Gramm *et al.* [57]. Next, we present a polynomial-time heuristic for the Closest String problem, and give an analysis that proves the algorithm is extremely efficient when the number of strings is relatively large. The key to obtaining this latter result is the application of smoothed analysis, an intermediate measure between average case analysis and worst case analysis; whereas average-case analysis studies the average behaviour of an algorithm over all instances of a problem, smoothed analysis studies the average behaviour of the algorithm on each "local region" of the instance space [103]. Section 4.1 of Chapter 4 is based on joint work with Stephane Durocher and Daniel G. Brown [15]. Section 4.2 is based on part of the joint work with Kathleen Wilkie [20].

In Chapter 4 we also consider the $O(n\ell + nd \cdot d^d)$-time algorithm by Gramm *et al.* [57], an algorithm that has been shown to work very efficiently in practice. We explain this good performance through smoothed analysis. Imperative to our analysis is the introduction of a perturbation model of the Closest String model and the probabilistic analysis of the $O(n\ell + nd \cdot d^d)$-time algorithm by Gramm *et al.* [57]. We show for any given Closest String instance, the average running time of this algorithm on a small perturbation of the instance is $O(n\ell + nd \cdot d^{2-\epsilon})$.

In Chapter 5 we investigate a slight augmentation of the Closest String problem that is a more realistic model of several biological problems. We give a systematic parameterized complexity analysis of this new model by considering several parameterizations of this problem. We give a negative complexity result by showing that the problem is W[1]-hard for unbounded alphabet size and every combination of a subset of the problem parameters. We also show that when the alphabet is unbounded, there exists a fixed parameter tractable algorithm with respect to two of the problem parameters. This chapter is based on joint work with Bin Ma [18].

In Chapter 6 we describe the application and implementation of the combinatorial and probabilistic insights described in the previous chapter to motif recognition. Applying these methods – as well as other insights – we develop sMCL-WMR, a program that is capable of detecting weak motifs in large datasets and is much faster than its predecessor MCL-WMR. Further, we show the capability of this program in detecting transcription factor binding sites in real biological data. In addition, we include work concerning canola genomics that was completed in collaboration with researchers at the University of Alberta and the Alberta Research Council. Canola exhibits several desirable nutritional and economic factors and therefore, is an important target for genomic research in Canada. By developing and employing genomic tools, which include sMCL-WMR and MCL-WMR, we are capable of identifying important regions of the canola genome that are responsible for specific biological activities. This knowledge may be used in the long-term aim of developing crop varieties with specific biological characteristics, such as being disease-resistant. The importance of these results is two-fold; it illustrates the capability of MCL-WMR and sMCL-WMR, and contributes to the study of crop genomics. This chapter is partly based on joint work with James King [17], and joint work with Limin Wu and Saleh Shah.

Finally, in Chapter 7 we conclude this thesis by outlining some related areas for future research. The areas extend from extensive and well-researched fields (such as the Closest String problem, and combinatorial methods for counting and sampling) to more interdisciplinary fields that require a careful search through diverse literature (such as finding biological applications to a variant of the Closest String problem).

In this rest of this chapter, we define the motif-recognition problem and several distinguishing string selection problems more formally, and motivate our study by giving several biological applications of these problems; one of these applications will be thoroughly explored in this chapter and Chapter 6.

1.1 From Transcription Factors to Pattern Recognition

From a high level, motif recognition can simply be described as classifying patterns (*a.k.a.* motifs) based on a priori knowledge or statistical information extracted from the data. Motif recognition is an important problem in biology; it and its variants have several important applications in bioinformatics. Before describing motif recognition from a computational perspective, we give an example motivating the investigation of this problem.

A *gene* is a region of DNA that codes for a type of protein or a RNA chain that has a function in the organism. Regulation genes, first discovered in 1961 by Jacob and Monod [62], are a type of gene that provides the instructions for creating proteins which help control the expression of other structural genes. Hence, we can think of a sequence of DNA coding for a particular protein that, due to the chemical properties of the amino acids it is made from, folds in a particular manner and so performs a particular function (*i.e.* enzymes, structural, regulatory). The protein encoding genes are regulated in the following three levels:

Transcription control level: determines if the transcription can begin.

Post-transcription control level: occurs after the DNA is transcribed and mRNA. Regulates how much the mRNA is translated into proteins by capping, splicing, and poly-adenylation. These processes occur in eukaryotes but not in prokaryotes. This modulation is a result of a protein or transcript which in turn is regulated and may have an affinity for certain sequences.

Post-translation control level: control on the protein level.

We adopt the convention that a DNA molecule is represented as a sequence whose symbols come from the four different nucleotides A, C, G, and T, and a protein is represented as a sequence whose symbols come from the 20 different amino acids.

All the above controls occur during distinct stages described in the *central dogma of molecular biology*, the axiom that genetic information flows from DNA to RNA to protein and and cannot flow in the reverse direction [33]. Figure 1.1 illustrates this process. The conversion from DNA to RNA is known as *transcription*, whereas, the decoding of a messenger RNA (mRNA) sequence into an amino acid sequence is referred to as *translation*. We will focus on transcription control.

A typical genome contains protein coding genes, non-coding RNAs, regulatory sequences, which we refer to as *promoters*, and regions that have no known function or are yet to be classified. In this section, we will restrict interest to the promoter regions,

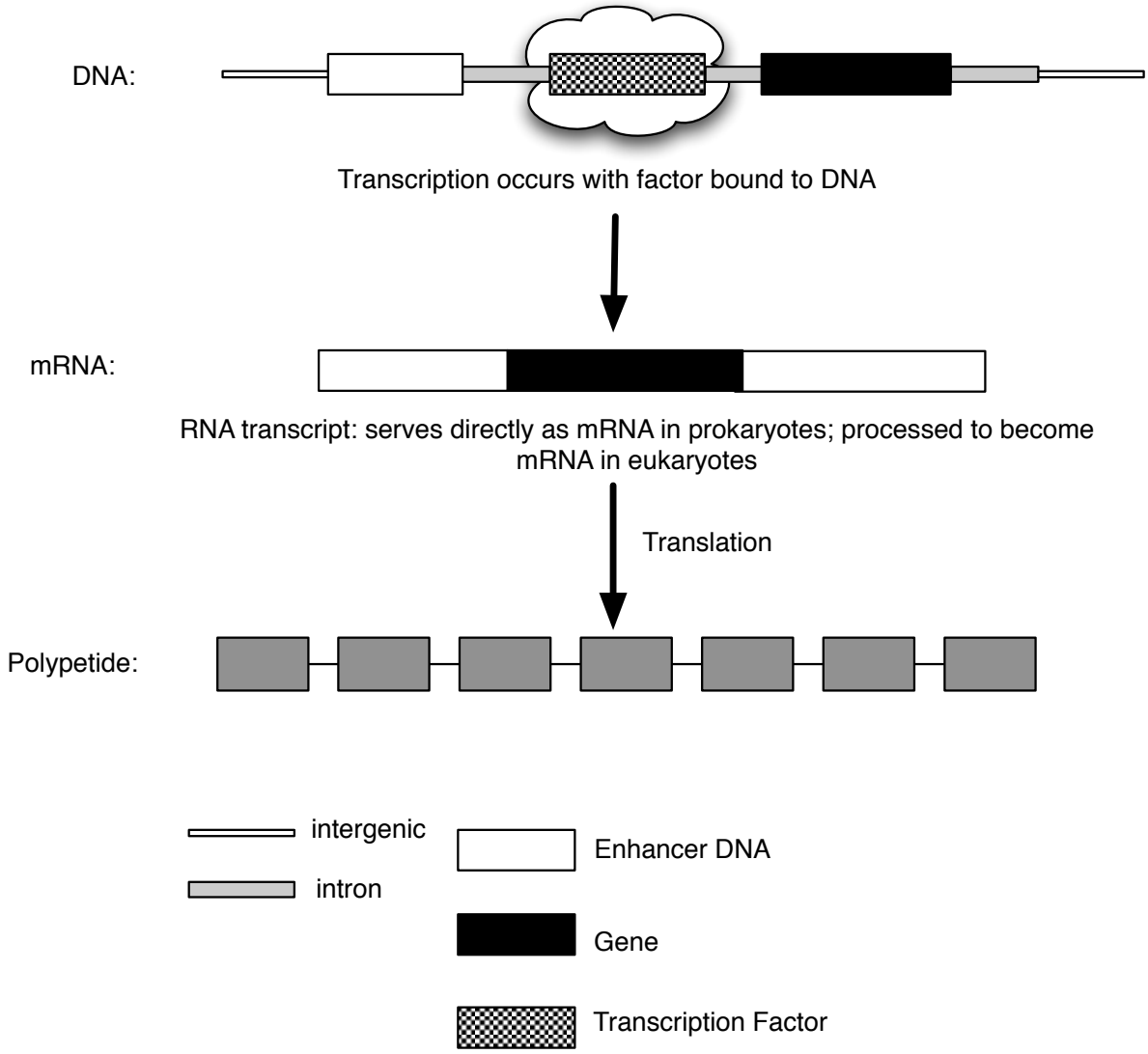

Figure 1.1: An illustration of the flow of genetic information from DNA to RNA to protein.

which are DNA sequences near the beginning of genes that signal RNA polymerase where to begin transcription.

The transcription process for a particular gene often requires that one or more transcription factors be bound to several specific regions, referred to as *binding sites*, that are located in the regulatory region of the gene. A single transcription factor can be bound to multiple binding sites; however they must have similar length and nucleotide pattern. This

specific nucleotide sequence will compromise the regions of interest and will encompass a motif in the context of motif recognition. Typically, a binding site is between five to twelve nucleotides (nt) in length but could be as large as 30 nt [12]. Note that nucleotide sequences are virtually synonymous with a sequence of base pairs (bp). Figure 1.2 illustrates a binding site with a bound transcription factor.

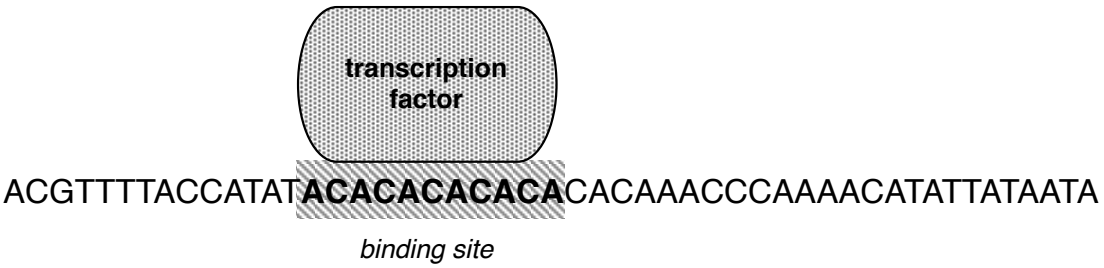

Figure 1.2: Depiction of a transcription factor binding site with bound transcription factor.

The discovery of motifs allows biologists to understand the complex mechanisms that regulate gene expression. A single transcription factor is often not sufficient for the regulation of transcription, and typically, there exists a set of transcription factors for a single gene. The binding sites usually occur as a recurring pattern in the sequence of nucleotides, however, due to the possibility of genetic mutation, a specific binding site in a nucleotide sequence will sometimes mismatch with the identified nucleotide pattern at a number of positions. Therefore, subsequences of a set of nucleotide data corresponding to the same binding site will likely not all match exactly, making the identification of these subsequences by computational means considerably more difficult.

The process of determining transcription factor binding sites can be described by the following steps:

1. Determine a set of promoters that contain the same binding site. This will be the input for the next step.

2. Identify reoccurring subsequences in the data that are likely are to be of biological interest using computational methods.

3. Experimentally evaluate the biological function of the subsequences identified in the previous step. Biological experiments to verify regulatory sites are tedious and time-consuming. One approach is to mutate different combinations of nucleotides until the functionality of the sequence changes.

We will briefly describe one method for the first of the steps in this process that uses co-expressed genes (genes that are expressed together). They are likely regulated by the same transcription factors and can be identified through clustering of microarray data. A microarray works by exploiting the ability of a given mRNA molecule to bind specifically to, or hybridize to, the DNA template from which it originated. An array containing many DNA samples can be used to determine (in a single experiment) the expression levels of hundreds or thousands of genes within a cell by measuring the amount of mRNA bound to each site on the array. With the aid of computational experiments, the amount of mRNA bound to the spots on the microarray is precisely measured, generating a profile of gene expression in the cell. Hence, using the microarray data we can find genes that are expected to be regulated by the same transcription factor, by measuring the level of transcription of mRNA based on presence or absence binding factors.

The second step listed above will be the main focus of this thesis. Identifying transcription factor binding sites is one specific example that will motivate our study of motif recognition. We will return to this specific biological problem later in this chapter, and in Chapter 6, but next we will more thoroughly describe motif recognition from a computational perspective.

1.2 The Motif-Recognition Problem

The bioinformatics literature uses disparate terminology and notation to define and describe the problem of identifying binding sites in regulatory regions. To facilitate our discussion, we begin with general definitions, then make these statements more formal when deemed necessary.

Given a number of biological sequences, motif recognition is the task of discovering meaningful patterns from the data without any prior knowledge. Consider a set of sequences $S = \{S_1, \ldots, S_n\}$. The aim is to use a computational algorithm to search for the common patterns in S, referred to as motifs, which are substrings of length ℓ, any two of which have at most d mismatches. It follows from this definition that a motif is a contiguous sequence that may or may not occur exactly in the input sequences due to the allowed degeneration. We will often use the term (ℓ, d)-*motif* to refer to a motif-recognition problem where the motif of interest has length ℓ and the degeneracy parameter is d. We will denote the length of s as $|s|$ and the jth letter of s as $s(j)$.

A motif is commonly described in one of two ways: by a center string, or by a *position weight matrix (PWM)*[1]. Given a number of length-ℓ sequences $S = \{s_1, \ldots, s_n\}$ and a parameter d, a *center string* is a length-ℓ string that has Hamming distance at most d

[1] Also called position-specific weight matrix (PSWM) or position-specific scoring matrix (PSSM) [11, 38]

from each sequence in S. Throughout this work, we will denote the Hamming distance between two strings s_i and s_j as $d(s_i, s_j)$, and denote the alphabet as Σ. We note that a center string is not necessarily unique for S. For example, if S contains n copies of a single length-ℓ string then there are $\sum_{i=0}^{d} \binom{\ell}{i}(|\Sigma| - 1)^i$ distinct center strings.

A PWM is a matrix of score values that gives a weighted match to any given substring of fixed length. It has one row for each symbol of the alphabet, and one column for each position in the pattern. The score assigned by a PWM to a substring $\overline{s} = \overline{s}(1)\overline{s}(2)\ldots\overline{s}(\ell)$ is defined as $\sum_{j=1}^{\ell} m_{\overline{s}(j),j}$, where j represents position in the substring, $\overline{s}(j)$ is the symbol at position j in the substring, and $m_{\alpha,j}$ is the score in row α, column j of the matrix. In other words, a PWM score is the sum of position-specific scores for each symbol in the substring.

A PWM assumes independence between positions in the input sequences, as it scores at each of the positions independently from the symbols at other positions. The score of a substring aligned with a PWM can be interpreted as the log-likelihood of the substring under a product multinomial distribution. Since each column defines log-likelihoods for each of the different symbols, where the sum of likelihoods in a column equals one, the PWM corresponds to a multinomial distribution. A PWM score is the sum of log-likelihoods, which corresponds to the product of likelihoods, meaning that the score of a PWM is then a product-multinomial distribution.

For example, given the following set of five strings (length 6, degeneracy parameter 1):

S_1 CACAGG
S_2 CACAGG
S_3 CACAGG
S_4 CCCAGG
S_5 AACAGG

the string CACAG clearly has Hamming distance at most 1 from each of these strings and thus is a center string. The position weight matrix for this set of strings is as follows:

	String position					
	1	2	3	4	5	6
A	1/5	4/5	0	1	0	0
C	4/5	1/5	1	0	0	0
G	0	0	0	0	1	1
T	0	0	0	0	0	0

The focus of our work will be on the following combinatorial formulation that was first introduced in 2000 by Pevzner and Sze [88]:

Definition (The Motif-Recognition Problem[2]) Let $S = \{S_1, \ldots, S_n\}$ be a set of sequences each of length m over the alphabet Σ, and s^* be the center string, a fixed and unknown sequence of length ℓ over the alphabet Σ. Suppose that s^* is contained in each S_i but is corrupted with at most d substitutions, so the Hamming distance of the occurrences from M is at most d. The aim is to determine s^* and the location of the motif instance in each sequence.

We note that this combinatorial definition restricts interest to a center sequence representation of a motif and hence, we will limit our focus to this motif representation. There exist several other variants of this problem definition [2, 48, 81, 92, 96]. For example, the Edited Motif Search problem [2, 92, 96] considers a database DB of sequences $S_1, S_2, \ldots, S_n$, and integers ℓ, d, and q. A solution to this problem consists of all the patterns in the DB such that each pattern is of length ℓ and occurs in at least q of the n sequences. A pattern U is considered an occurrence of another pattern V if the edit distance between U and V is at most d.

Throughout this thesis, we will also consider the optimization version of this problem. Where there exists ambiguity, we will explicitly indicate whether we are considering the optimization or decision version of the motif-recognition problem.

Definition (The Motif-Recognition Optimization Problem) Let $S = \{S_1, \ldots, S_n\}$ be a set of sequences, each of length m over the alphabet Σ, and M be the center string, a fixed and unknown sequence of length ℓ over the alphabet Σ. The aim is to find a length ℓ substring s_i in each string of S_i and string s of length ℓ over Σ minimizing d_c, where $d(s, s_i) \leq d_c$.

1.3 String Selection Problems

String selection and comparison problems belong to the more general class of problems in bioinformatics where a finite set of strings is given and the aim is to determine their *center string*. The idea of the center string can be related to several different objectives, including the following:

[2]This problem is also referred to as the Closest Substring problem [67, 75] and as the Common Approximate Substring problem [102]

- the objective of the problem is to determine the center string that minimizes the maximum distance from each input string (CLOSEST STRING problem);
- the objective of the problem is to determine the center string that maximizes the minimum distance from each input string (FARTHEST STRING problem).

1.3.1 The CLOSEST STRING Problem

Due to its relation to motif recognition and other topics in biology, the CLOSEST STRING problem and its variants have been studied extensively in bioinformatics and computational biology [22, 23, 57, 67, 70, 77, 78, 109]. According to Gramm *et al.* the CLOSEST STRING problem "is one of the core problems in the field of consensus word analysis with particular importance for computational biology" [57].

Definition (The Closest String Problem[3]) Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of sequences, each of length-ℓ and over an alphabet Σ, and a parameter d, find a string s of length ℓ over Σ such that for every string $s_i \in S$ $d(s, s_i) \leq d$.

We refer to s in the previous definition as a *center string*.

CLOSEST STRING is NP-complete, even for binary sequences; this was first shown by Frances and Litman [49] by considering an equivalent problem in coding theory. Therefore, no polynomial-time solution is possible unless $P = NP$.

s1	ACCCTACACTG	s1	ACCCTACACTG
s2	CACCTACACTG	s2	CACCTACACTG
s3	CCCCTACACTG	s3	CCCCTACACTG
s4	AACCTACACTG	s4	ACCCTACACTG
s5	AACCTACACTG	s5	ACCCTACACTG
s6	CCCCTACACTG	s6	CCCCTACACTG
			ACCCTACACTG

Figure 1.3: An example showing two different CLOSEST STRING instances; one that is a motif set (right) and one that is a decoy set (left).

[3]or equivalently, the CONSENSUS STRING problem or the CENTER STRING problem

Clearly, for a set S to have at least one center string corresponding to the degeneracy parameter d, the Hamming distance between any pair of strings in S must not exceed $2d$. We refer to such a set of strings as being *pairwise bounded*. Determining whether a set of strings is pairwise bounded can be trivially decided in polynomial time and therefore, since the CLOSEST STRING problem is NP-complete, there must exist sets of strings that are pairwise bounded but do not contain a center string, unless P = NP. In figure 1.3 there exist two sets of strings that are pairwise bounded when $d = 1$, however, the set on the left does not contain a length-ℓ string that has distance at most d from each string in the set and the string on the right has such a string (*i.e.* the string in bold). Thus, the CLOSEST STRING problem essentially reduces to discerning between pairwise bounded sets that have a center string (and if so, finding one such string) and those sets that do not. A set of strings S is a *motif set* if there exists a center string and is a *decoy set* if S is pairwise bounded but does not have a center string.

Also of considerable importance is the optimization version of the CLOSEST STRING problem. Throughout this work, we will explicitly indicate when there exists ambiguity whether we are considering the optimization or decision version of the CLOSEST STRING problem.

Definition (The Closest String Optimization Problem) Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of sequences, each of length ℓ and over an alphabet Σ, find a string s of length ℓ over Σ minimizing d_c such that, for every string $s_i \in S$, $d(s, s_i) \leq d_c$.

We refer to the string s in the context of the optimization version on the problem as the *center string*.

1.3.2 The FARTHEST STRING Problem

The FARTHEST STRING problem was first introduced by Lanctot *et al.* [66]. Whereas, the CLOSEST STRING problem abstractly defines the problem of finding a pattern that, with some error, occurs in one set of strings, the FARTHEST STRING problem defines the problem where the pattern does not occur in a set of strings. Both of these string selection problems have application to the analysis of genomic data. The FARTHEST STRING problem has been proved NP-complete even when the alphabet is binary [66], and therefore, it is unlikely to have an exact polynomial-time solution, unless P = NP. This problem can be more formally defined as follows:

Definition (The Farthest String Problem) Given a set S_f of strings of length at least ℓ over an alphabet Σ and a non-negative parameter d_f, the objective is to determine if there exists a string s over the alphabet Σ such that for any $s_i \in S_f$, $d(s, s_i) \geq d_f$.

Although the Farthest String problem is not as well-studied as the Closest String problem, it will be of both theoretical and practical interest in this thesis.

1.4 Biological Applications

We introduced motif recognition by demonstrating its applicability to detecting transcription factor binding sites; however, there exist many other biological and non-biological applications for this problem. For example, motif recognition is applicable in the fields of coding theory [32, 49] and data compression [53]. There exist many other biological applications in addition to the ones discussed in this section, including finding an unbiased consensus of a protein family [10], function prediction [58, 102], and modelling and predicting splice sites [8].

Designing Diagnostic Probes

Creating diagnostic probes for bacterial infection has been one of the core applications to the theoretical study of string selection and comparison problems [10, 67, 79]. Currently, oligonucleotide microarrays are being used in gene expression analysis, as well as for diagnostic purposes (*i.e.* the identification of micro-organisms in clinical and environmental samples). The key task is to efficiently determine suitable sets of oligonucleotide probes that can reliably detect and differentiate the target sequences. Determining efficient algorithms that achieve this are of utmost importance since the datasets may be significantly large. There exist algorithms to produce adequate probes when there exists a high amount of variability within the target sequences [64, 65]; however, when the sequence database contains homologous genes, the problem still remains largely unsolved. In the case where it is impossible to determine specific probes due to the high similarity, it is advantageous to design probes that are specific for groups of closely related sequences, and that detect target sequences as well as some non-target sequences (referred to as negative probes). Hence, given a set of DNA sequences from a group of closely related pathogenic bacteria and a host, the task is to find a sequence that occurs in each of the bacterial sequences without occurring in the host sequence.

Polymerase Chain Reaction

Polymerase chain reaction (PCR) is a well-established technique in molecular biology to amplify a single or few copies of a DNA segment, referred to as a template, across several orders of magnitude, generating thousands to millions of copies of a particular DNA sequence. *Primers* are short DNA fragments that contain sequences complementary to the

target region along with a DNA polymerase. Hence, they are key components which enable selective and repeated amplification. The DNA generated is itself used as a template for replication during the PCR by setting a chain reaction in which the DNA template is exponentially amplified. PCR can be modified to perform a wide array of genetic manipulations [91].

Creating universal PCR primers that are able to recognize multiple segments has been a problem well-investigated by the bioinformatics community [39, 61, 67, 74, 91]. The specificity of the primer hybridization directly affects the specificity of the amplification by PCR. Designing well-constructed primers comes down to a sequence problem where one attempts to determine the maximum number of mismatches that can be allowed for hybridization.

Drug Design

Drug design is another biological application [34, 36, 67]. Given a set of sequences of orthologous genes from a group of closely related pathogens, and a host, the goal is to find a subsequence that is highly conserved in all of the pathogen sequences but not conserved in the sequence of the host. This subsequence can in turn, be used to create a novel drug that harms several pathogens with minimal effect on the host.

1.5 Summary

This thesis is focused on the development of combinatorial and probabilistic algorithms for problems arising from the analysis of genomic data. In particular, we will examine several traditional problems in bioinformatics, including motif recognition, Closest String, and other string selection problems. In this chapter we introduced each of these problems and motivated them by illustrating their application in biology. Next, we will survey some of the important results through the literature pertinent to motif recognition, Closest String, and its variants.

Chapter 2

Related Work

The contributions of this thesis lie at the cusp between discrete mathematics and bioinformatics. This work involves constructing predictive models to identify important components of genetic sequence data. Modelling biological problems as graphs and other abstract mathematical objects can lead to theoretical results concerning computational complexity and, thus, the ability to find an approximate solution efficiently. In Chapter 1 we introduced the central topic of this thesis, motif recognition, gave biological motivation for the study of this problem, and defined several string selection problems peripheral to this investigation.

In this chapter, we begin by surveying the theoretical results related to motif-recognition, the Closest String problem and variants of the Closest String problem. Each of these problems that we consider is NP-complete and, thus, unlikely to have a polynomial-time algorithm. Two natural approaches to dealing with the intrinsic computational hardness of these sequence problems are: to consider their parameterized complexity, and to determine if they can be approximated to a reasonable factor in polynomial time. Prior to this survey, we introduce and define terms from theoretical computer science that will arise in this work, namely those central to the topics of parameterized complexity and approximability.

In addition to our survey on the important theoretical results related to our study of motif recognition, we give an overview of a short list of programs that detect motifs in synthetic and biological data. The programs we choose to survey are the ones most well-used or well-studied; the work is either well-cited or frequently used by biologists or bioinformaticians.

2.1 Preliminaries

We provide an overview of the terms and definitions used in this thesis, and a detailed review of parameterized complexity and the theory of approximation algorithms.

2.1.1 Notation

Let s be a string over the alphabet Σ. Denote the length of s by $|s|$, and the jth letter of s by $s(j)$. Hence, $s = s(1)s(2)\ldots s(|s|)$.

Given functions f and g of a natural number variable n, the notation $f \asymp g$ $(n \to \infty)$ is used to express that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$$

and f is an asymptotic estimation of g (for relatively large values of n). Throughout this work, we will often refer to a short nucleotide sequence of a specific length k as a *k-mer*.

Definition (closest string) Given a set of strings $S = \{s_1, \ldots, s_n\}$, each string of length ℓ, then a string s is a closest string for S if and only if there is no string s' such that $\max_{i=1,\ldots,n} d(s', s_i) < \max_{i=1,\ldots,n} d(s, s_i)$.

Similarly, if s is a closest string for S then we define the *optimal closest distance* d is equal to $\max_{i=1,\ldots,n} d(s, s_i)$.

Definition (majority vote string) We refer to a majority vote string for S as a length-ℓ string containing a letter that occurs most often at each position. A majority vote string – which we sometimes refer to as the *majority string* – is not necessarily unique.

There can be up to $|\Sigma|$ unique majority symbols at each position, and up to $|\Sigma|^\ell$ distinct majority strings. When we refer to a randomly selected majority vote string we refer to a string that has the majority symbol at each position with ties broken arbitrarily.

2.1.2 A Brief Introduction to Parameterized Complexity

Any NP-hard problem Φ is unlikely to be solved in polynomial time. Most likely there will be only exponential-time algorithms for Φ. Fortunately, an exponential-time algorithm can still be efficient if the exponential component of the running time is restricted to parameters that are likely small in practice and the running time is polynomial in all other parameters. The goal of parameterized complexity is to attempt to restrict the exponential increase of the running time to as few parameters of the instance as possible.

Definition A *parameterized problem* is a language L contained in $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. The second component is called the *parameter* of the problem.

A parameterized problem is specified by three pieces of information: the input, the problem, and the parameters that are designated as fixed. In classical complexity, tractable problems are defined as those problems solvable by a polynomial-time algorithm. The analogous concept in parameterized complexity is an algorithm with running time bounded by a function that is polynomial in the size of the input but is allowed to be superpolynomial in the value of the fixed parameters.

Definition A problem φ is said to be *fixed parameter tractable* with respect to parameter k if there exists an algorithm that solves φ in $f(k) \cdot n^{O(1)}$ time, where f is a function of k that is independent of n.

For example, given a graph $G = (V, E)$ with vertex set V, edge set E, and positive integer k, the VERTEX COVER problem aims to discern where there is a subset of vertices $V_C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in V_C. The VERTEX COVER problem is NP-complete [51] but is fixed parameter tractable since there exist algorithmic solutions that have running time $O(kn + 1.3^k)$ [40]. There are several general, sophisticated techniques developed for the design of efficient parameterized algorithms, including the bounded search tree method, reduction to the problem kernel, and perfect hashing [40]. A problem that is fixed parameter tractable is said to reside in the parameterized corresponding complexity class FPT.

Not all problems in NP are believed to be in FPT. For example, consider the NP-complete CLIQUE problem: given an undirected graph $G = (V, E)$ and a positive integer k, the aim is to determine whether there is a subset of vertices $C \subseteq V$ of size at least k where each pair of vertices in C are connected by an edge. CLIQUE is believed to be *fixed parameter intractable* since it is not known whether it can be solved in time $f(k) \cdot n^{O(1)}$, where f might be an arbitrarily fast growing function only depending on k. The best known algorithms for solving clique runs in time $O(n^{o(k)})$ [40].

In order to characterize those problems that do not seem to admit a fixed parameter efficient algorithm, Downey and Fellows [40] defined a *fixed parameter reduction*. Let $L, L^* \subseteq \sum^* \times \mathbb{N}$ be two parameterized languages. L *reduces* to L' if there are functions $k \to k'$ and $k \to k''$ from $\mathbb{N}$ to $\mathbb{N}$ and a function $(x, k) \to x'$ from $\sum^* \times \mathbb{N}$ to $\sum^*$ such that:

1. $(x, k) \to x'$ is computable in time $k''|x|^c$, for some constant c and
2. $(x, k) \in L$ if and only if $(x', k') \in L'$.

We note that if L reduces to L' and L' is FPT, then L is FPT.

The goal of parameterized complexity is to determine a finer distinction between problems lying in NP and organize them according to some hierarchy of classes. The central problem used to provide a model for parameterized non-deterministic computation is the Weighted Circuit Satisfiability problem. The parameterized version of this problem defines an infinite hierarchy of parameterized complexity classes inside NP.

Definition A *Boolean circuit* α_n with input $x = x_1x_2\cdots x_n$ of length n is a directed acyclic graph. The nodes of fan-in 0 are called *input nodes* and are labelled from the set $\{0, 1, x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_n, \overline{x_n}\}$. The nodes of fan-in greater than 0 are called *gates* and are labelled either AND, OR, NOT. A special node is designated the output node. The *size* is the number of nodes and the *depth* is the maximum distance from an input node to the output node.

Definition A gate is said to be *large* if the number of inputs to that gate exceeds some specified bound. The *weft* of a decision circuit is the maximum number of large gates on any path from the input variables to the output.

Definition Given a weft t depth h decision circuit C and a parameter k, the Weighted Weft t Depth h Circuit Satisfiability ($\mathrm{WCS}_{t,h}$) problem aims to determine whether C has a weight k satisfying assignment.

The set of problems reducible under parameterized reductions to $\mathrm{WCS}_{t,h}$ for any h, forms the class called $W[t]$. The W hierarchy is a collection of computational complexity classes used in the theory of parameterized complexity, classifying computational problems according to their apparent intractability in terms of a parameter other than input size.

In this thesis we will mainly be interested in the W[1] class, which is considered the lowest intractable class of parameterized complexity. From a practical perspective, W[1]-hardness gives a concrete indication that a parameterized problem with parameter k is unlikely to have an algorithm that has running time of the form $f(k) \cdot n^{O(1)}$. XP is the class of all problems solvable in $O(n^{f(k)})$, where n is the size of the input, k a parameter, and f is a computable function independent of n.

The parameterized complexity classes are related to P and NP as follows:

$$\mathrm{P} \subseteq \mathrm{FPT} \subseteq \mathrm{W[1]} \subseteq \mathrm{W[2]} \subseteq \cdots \subseteq \mathrm{W[P]} \subseteq \cdots \subseteq \mathrm{XP} \subseteq \mathrm{NP}.$$

Parameterized complexity is based on the assumption that FPT $\neq$ W[1], which is very much analogous to the conjecture that P$\neq$NP [40].

2.1.3 A Brief Introduction to Approximability

Approximation algorithms are used to find approximate solutions to optimization problems and are often associated with NP-hard problems. Since it is unlikely that there can be an efficient, polynomial-time algorithm for solving a NP-hard problem exactly, it is useful to find a reasonable – though sub-optimal – solution in polynomial time. Approximation theory was first introduced by Johnson [63] and since this time and has become a widely-accepted manner to deal with the tractability of hard problems. When discussing approximability, the efficiency of the algorithm, the correctness of the solution, and the quality of the approximation are all important concerns. Various measures of approximation quality have been proposed, however, the *performance ratio* will be our focus throughout this thesis.

Definition Let x be an instance of an optimization problem π having an optimal solution $opt(x)$. Let A be an algorithm for solving π, and $A(x)$ the value of the solution produced by A when applied to x. The performance ratio of A with respect to x is

$$\max\{\frac{A(x)}{opt(x)}, \frac{opt(x)}{A(x)}\}.$$

Many results prove the approximation is optimal up to a small constant factor. For example, a ρ-approximation algorithm $\mathcal{A}$ for any input x will return a solution whose value will be within a factor ρ times the optimal solution. For example, the following polynomial-time algorithm is a 2-approximation algorithm for the vertex cover problem: find an uncovered edge and add both endpoints to the vertex cover, until no uncovered edge remains. It is clear that the resulting set of vertices is a vertex cover and is at most twice as large as the optimal one.

Different optimization problems have different approximation properties relating to the types of approximation algorithms for solving them. In a ρ-approximation algorithm, the value of ρ may be be a function of $|x|$, some other parameter to the problem, or a constant. A *polynomial-time approximation scheme (PTAS)* is an algorithm which takes an instance of an optimization problem and a parameter $\epsilon > 0$ and, in polynomial time, produces a solution that is within a factor ϵ of being optimal. Hence, the existence of a PTAS implies that the problem can be approximated within an arbitrarily small factor in polynomial-time. Not all problems exhibit a PTAS; in fact, any problem hard for the class known as APX does not have a PTAS, unless P=NP [113]. We note that the running time of a PTAS is required to be polynomial in the size of the problem instance for every fixed ϵ but can differ for various values of ϵ.

An *efficient PTAS (EPTAS)* is an approximation scheme that produces a $(1 + \epsilon)$-approximation in $f(\epsilon) \cdot n^c$ time for some constant c. For example, if $f(\epsilon)$ is $2^{1/\epsilon}$ then such an approximation scheme can be practical even for $\epsilon = 0.1$ and large n. A consequence of W[1]-hardness is that there is no EPTAS for the optimization version of the problem [26].

2.1.4 A Brief Introduction to Smoothed Analysis

Smoothed analysis was introduced as an intermediate measure between worst-case and average-case analysis, and is used to explain how many algorithms that are deemed inefficient by worst-case analysis can efficiently find good solutions in practice. It works by showing that the worst-case instances are fragile to small change; slightly perturbing a worst-case instance destroys the property of it being worst-case. Spielman and Teng [103] analytically showed that the smoothed complexity of the simplex algorithm (with the shadow-vertex pivot rule) for linear programming is a polynomial-time algorithm in practice. Linear programming is a continuous problem. The input is a sequence of real numbers: a cost vector and linear constraints. The smoothing operation adds Gaussian noise with parameter σ to each number in the input. The expected running time of the simplex algorithm for such a perturbed instance is polynomial in $1/\sigma$ and the number of input variables.

Several other papers discuss the smoothed complexity of continuous and discrete problems [13, 7, 76]. Banderier *et al.* [7] investigated the concept of smoothed analysis with respect to ordering problems. They presented and used a *partial permutations* perturbation model, where each element of the sequence is independently selected with a given probability p and then a random permutation on the selected elements is performed. Among other results, Banderier *et al.* [7] proved that the expected running time of quicksort on a partial permutation of n elements is $O\left(\frac{n}{p \cdot \log n}\right)$. Manthey and Reischuk investigated the smoothed analysis of binary search trees and prove tight lower and upper bounds of roughly $\Theta(\sqrt{n})$ for the expected height of a binary search tree.

The smoothed complexity of other string and sequence problems has been considered by Andoni and Krauthgamer [5], Manthey and Reischuk [80], and Ma [76]. Andoni and Krauthgamer [5] studied the smoothed complexity of sequence alignment by the use of a novel model of edit distance. Their results demonstrate the efficiency of several tools used for sequence alignment, most notably PatternHunter [78]. Manthey and Reischuk gave several results considering the smoothed analysis of binary search trees [80]. Ma demonstrated that a simple greedy algorithm runs efficiently in practice for Shortest Common Superstring [76], a problem that has application to string compression and DNA sequence assembly.

2.2 Past Work on Motif Recognition

We have previously introduced the problem of motif recognition and showed the applicability to finding transcription factor binding sites. An obvious method to detect motif instances of length ℓ is to generate all possible sequences of length ℓ and examine each of

these sequences to see if there exists a subsequence in the input that has distance at most d from it. There are $|\Sigma|^\ell$ possible center sequences and each of these sequences can be checked in time $O(mn\ell)$ time. This is essentially one of the first motif-recognition algorithms given by Waterman *et al.* [116]. Although this exhaustive algorithm is guaranteed to find a motif set if there exists one, the running time of the algorithm is exorbitantly large and, hence, impractical for even reasonably small motifs. Nonetheless, due to the applicability of the problem to important problems in biology, it is imperative that efficient algorithms be developed for it. Numerous algorithms have been developed to solve specific instances of the problem, including PROJECTION [24], Winnower [88], pattern driven approaches [107], MITRA [43], PSM1 [92], PMSprune [35], the Voting algorithm [30], MEME [6], VAS [31], RISOTTO [89], Weeder [87] and several others. We will discuss several of these algorithms in more detail in this section.

2.2.1 Theoretical Results

A crucial first step in the study of the complexity of motif recognition is the determination of whether the problem lies within the complexity class NP. The *Verify Algorithm* demonstrates that the problem can be solved in non-deterministic polynomial time. The algorithm aligns the supposed center string with all possible length-ℓ substrings in the set of the input sequences, counts the number of matching positions, and returns whether or not it is a valid center string. Hence, the complexity of the algorithm is $\Theta(nm\ell)$.

Algorithm 1 Verify Algorithm

Input: A set $S = \{S_1, S_2, \ldots, S_n\}$ of n length m sequences, a string $s \in \Sigma^\ell$, and parameter d.
Output: A boolean value indicating whether there is a length ℓ substring in each sequence in S with distance at most d from s.
Let $\alpha \leftarrow$ true.
For $i = 1, \ldots, m$:
    Let $\alpha_i \leftarrow$ false.
    For each length ℓ substring s_i of S_i:
        If $d(s_i, s) \leq d$ **then** $\alpha_i =$ true.
    If α_i is false **then** $\alpha =$ false.
Return α.

After the verification that the problem is in NP, the next aim is to determine whether it is in P or NP-complete. Unfortunately, the problem is known to be NP-complete, and unlikely to be solved in polynomial-time, unless P = NP [49]. There are two natural approaches to investigate the computational intractability of this problem: consider the

fixed parameter tractability with respect to the problem parameters (*i.e.* ℓ, n, or d), and consider the ease with which it can be approximated with reasonable accuracy and efficiency.

In [44] and [45] it is shown that the problem is W[1]-hard even if all three of n, d and ℓ are fixed and the alphabet is unbounded. Smith demonstrated the problem is W[1]-hard when m and ℓ are parameters and $|\Sigma|$ is unbounded [102]. On the other hand, if $|\Sigma|$ and ℓ are both parameters, then the problem becomes fixed-parameter tractable since we can enumerate and check all the $|\Sigma|^{\ell}$ possible center strings.

If the strings are long, which is often the case in practical applications, then it makes more sense to assume that the number of sequences n or the degeneracy d are parameters. In [45] it is shown that the problem is W[1]-hard with parameter n, even if the alphabet is binary. Recently, Marx [82] demonstrated that motif recognition is W[1]-hard with respect to the parameter d or with the combined parameters d and n, even if the alphabet is binary. Table 2.1 summarizes these results. Further, Marx [82] presented two algorithms that aim to be efficient for small fixed values of d and n: for some functions f and g and size of the input k, the algorithms have running times $f(d) \cdot k^{O(\log d)}$ and $f(d,n) \cdot k^{O(\log \log n)}$, respectively. The second algorithm is based on connections with extremal combinatorics of hypergraphs.

Lastly, Ma and Sun [77] gave a new algorithm with improved time complexity $O((16|\Sigma|)^{d} \cdot nm^{\lceil \log d \rceil + 1})$. Smith gave some results considering membership in the W-Hierarchy beyond the W[1] parameterized complexity class [102]; however, we will restrict interest to the W[1]-hardness throughout this thesis. In addition, he presented the first fixed parameter tractable variant not parameterized with the alphabet size [102].

Parameter(s)	$\|\Sigma\|$ is constant	$\|\Sigma\|$ is a parameter	$\|\Sigma\|$ is unbounded
d	W[1]-hard [45]	W[1]-hard [45]	W[1]-hard [44, 45]
d, n	W[1]-hard [45]	W[1]-hard [45]	W[1]-hard [44, 45]
n	W[1]-hard [45]	W[1]-hard [45]	W[1]-hard [44, 45]
ℓ	FPT (trivial)	FPT (trivial)	W[1]-hard [44, 45]
ℓ, d, n	FPT (trivial)	FPT (trivial)	W[1]-hard [44, 45]
ℓ, m	FPT (trivial)	FPT (trivial)	W[1]-hard [102]

Table 2.1: Complexity of motif recognition with different parameterizations

There is a straightforward 2-approximation algorithm for the optimization version of the motif-recognition problem. Li *et al.* [70] presented a polynomial-time algorithm that achieves a $2 - 2/(2|\Sigma| + 1)$-approximation guarantee for this problem. Lanctot *et al.* [67] improved upon this result by giving an algorithm that achieves a $(4/3 + \epsilon)$-approximation guarantee in polynomial time, for any small constant $\epsilon > 0$. Ma [75] presented a PTAS for

the problem. For a given value of r, this PTAS considers all choices of r substrings of length ℓ from the n strings and for each collection of substrings chooses a majority string as the center string. After all $n^r(m-\ell+1)^r$ possible collections of substrings are considered, the collection with the smallest maximum (Hamming) distance between the center string and any string in the collection. Smith [102] gave a PTAS for the optimization version of the motif-recognition problem where there are no restrictions on the alphabet size, and gave an improvement to the result of Ma [75] for the case where the alphabet is binary. Andoni *et al.* [4] presented a PTAS that achieves a much better time complexity of $O\left(mn^{O(\epsilon^{-2}\log\frac{1}{\epsilon})}\right)$. Lastly, Ma and Sun [77] combined their fixed parameter tractability results to provide a simpler PTAS with time complexity $O\left(mn^{O(\epsilon^{-2})}\right)$. As suggested by Ma and Sun [77], an important problem that remains open is to find a more efficient approximation algorithm for $\log n < d < \log n/\epsilon^2$.

In addition, Smith [102] presented the idea that the motif recognition optimization problem can be optimized with respect to different objectives, rather than just minimizing d. The different objectives they described are:

- Maximize $\ell - d$, which described the similarity between the strings (Max Closest Substring);
- Maximize ℓ while keeping d constant (Longest Common Approximate Substring);
- Maximize n, the number of strings (Maximum Coverage Approximate Substring).

Smith proved that Max Closest Substring is not approximable within $\log n/4$ in polynomial time (unless P=NP) through a gap preserving reduction the Set Cover problem. He also proved Maximum Coverage Approximate Substring is APX-hard, and finally, that Longest Common Approximate Substring cannot be approximated in polynomial time with performance ratio better than $2-\epsilon$, for any $\epsilon > 0$ unless P=NP [102]. He gave a 2-approximation algorithm for Longest Common Approximate Substring and a $|\Sigma|^d$-approximation algorithm for Maximum Coverage Approximate Substring.

Deng *et al.* [36] considered a related string problem, referred to as the Distinguishing (Sub)string Selection problem. Given two sets of strings, S_b (bad genes) and S_g (good genes), and two integers d_b and d_g ($d_b \leq d_g$), the aim of the Distinguishing (Sub)string Selection problem is to find a (distinguishing) substring s of length ℓ that distinguishes the bad strings from good strings – that is, for each string $S_i \in S_b$ there exists a length-ℓ substring s_i of S_i with $d(s, s_i) \leq d_b$ (close to bad strings) and for every substring t_i of

length-ℓ of every string $T_i \in S_b$, $d(s, t_i) \geq d_g$ (far from good strings). Deng *et al.* [36] presented a PTAS for this problem. Gramm *et al.* [54, 55] showed the decision version of this problem is W[1]-hard with respect to all combinations of the parameters as given in the problem definition, and gave a fixed parameter algorithm for a very restricted version of the Distinguishing (Sub)string Selection problem.

2.2.2 Statistical Methods

We discuss two popular statistical methods for detecting motifs: Multiple EM for Motif Elicitation (MEME) [6] and Gibbs sampling [68]. Other notable statistical methods include CONSENSUS [60], the Yeast Motif Detection (YMF) approach [101] and Motif Discovery Scan (MDscan) [73]. Unless otherwise stated, we assume the input to a motif-recognition program is a set of n sequences of length m denoted as $\{S_1, S_2, \ldots, S_n\}$ and the aim is to return is to return a set of length-ℓ sequences $\{s_1, s_2, \ldots, s_n\}$, where s_i is a subsequence of S_i.

MEME

An *Expectation Maximization (EM)* algorithm is a method for finding maximum likelihood estimates of parameters in statistical models, where the model depends on unobserved latent variables. It is an iterative algorithm that alternates between performing an *expectation step*, which computes the expectation of the log-likelihood evaluated using the current estimate for the latent variables, and a *maximization step*, which computes parameters maximizing the expected log-likelihood found on the expectation step. These estimates of the parameters are used to determine the distribution of the latent variables in the next expectation step [37].

EM can be used to simultaneously optimize the PWM description of a motif [37]. The weight matrix for the motif is initialized with a subsequence of length ℓ, plus a small amount of background nucleotide frequencies, for each subsequence of length ℓ in the target sequences the probability that it was generated by the motif is calculated (rather than by the background sequence distribution). EM takes a weighted average across these probabilities to generate a more refined motif model. The algorithm iterates between calculating the probability of each site based on the current motif model, and calculating a new motif model based on the probabilities. It can be shown that this procedure performs a gradient descent, converging to a maximum of the log likelihood of the resulting model.

Developed in 1995 by Bailey and Elkan, MEME [6] is one of the most popular motif-recognition applications. The idea is to find an initial motif, then repeat the expectation and maximization steps to improve the motif until it cannot be improved beyond a certain

threshold, or the maximum number of iterations has been reached. MEME uses the PWM motif representation.

The algorithm has a parameter γ that denotes the probability that a sequence is a motif, a motif model θ that is used to represent the motif, and a background model θ_0 that is used to represent the distribution of the sequences involved. Both the motif model and the background model are represented by a PWM. The expectation step computes the probability of finding the site at every position in every sequence based on the given PWM, and in the maximization step, we refine the PWM of the motif given the probabilities for every position and every sequence. We refine the PWM for θ by letting the PWM be equal to the probability that a subsequence has a specific nucleotide at a given position. These probabilities are then normalized so that the entries for a column of the PWM sum to one. Further, with the probability we get from the expectation step, we update the value of γ.

For example, suppose we are given the sequence $s = TGATATAACGATC$, and the following PWMs for θ and θ_0:

θ	1	2	3	4	5
A	0.2	0.8	0.1	0.7	0.8
C	0	0.1	0.2	0	0.1
G	0.1	0.1	0.1	0.2	0.1
T	0.7	0	0.6	0.1	0

θ_0	1	2	3	4	5
A	0.25	0.25	0.25	0.25	0.25
C	0.25	0.25	0.25	0.25	0.25
G	0.25	0.25	0.25	0.25	0.25
T	0.25	0.25	0.25	0.25	0.25

As in the expectation step, we consider the subsequence at each possible position of s. Let p_i be the probability that of finding the given subsequence of length 5 at the given position. Hence, there are 9 possible subsequences and probabilities to consider.

```
TGATATAACGATC
TGATA          p1
 GATAT         p2
  ATATA        p3
   TATAA       p4
    ATAAC      p5
     TAACG     p6
      AACGA    p7
       ACGAT   p8
        CGATC  p9
```

Considering the first segment $TGATA$, we calculate the probability p_1 as follows:

$$p_1 = \frac{\Pr(TGATA|\theta)\gamma}{\Pr(TGATA|\theta)\gamma + \Pr(TGATA|\theta)(1-\gamma)}.$$

With the probabilities that we get from the expectation set, we calculate γ to be equal to $\sum_{\forall i} p_i/9$ and refine the PWM θ. *e.g.* the entry for the nucleotide T at position 1 of the refined PWM θ is equal to:

$$\frac{p_1 + p_4 + p_6}{\sum_{\forall i} p_i}.$$

The MEME algorithm has several drawbacks, including the following: erased input data each time a new motif is discovered since it assumes the new motif is correct, limitation to the two-component case, and extreme pessimism about alignment (which might lead to missed signals). The most significant drawback of the algorithm is its high time complexity.

Gibbs Sampler

In 1993 Lawerence *et al.* [68] presented GibbsDNA, a motif-recognition program that uses a randomized approach to iteratively improve a motif. Gibbs sampling can be viewed as a stochastic implementation of EM. Whereas the latter takes a weighted average across all subsequences (weighted with the current estimate of the probability that they belong to the motif), Gibbs sampling takes a weighted sample from these subsequences. Initially, a motif, represented by a PWM, is selected by choosing one length-ℓ subsequence uniformly at random from each of the input sequences. Let θ be the motif model, and θ_0 as the distribution of the sequences. Figure 2.1 illustrates this initial step of Gibbs sampling.

At each iteration, the algorithm probabilistically decides whether to add a new site and/or remove an old site from the motif model, weighted by the binding probability for those sites. The resulting motif model is then updated, and the binding probabilities recalculated. Given sufficient iterations, the algorithm will efficiently sample the joint probability distribution of motif models and sites assigned to the motif, focusing in on the best-fitting combinations. The algorithm repeatedly performs the following steps to iteratively improve the motif:

1. It selects a sequence S_i at random from the set of input sequences and deletes its length-ℓ subsequence.

2. Next, it defines the PWM θ based on the remaining length-ℓ sequences (*i.e.* the subsequences in bold in Figure 2.2), and the PWM θ_0 based on the non-motif regions.

3. For each length-ℓ subsequence s_{ij} from sequence S_i and starting at position j, it calculates $\frac{\Pr(s_{ij}|\theta)}{\Pr(s_{ij}|\theta_0)}$, which is denoted as $score_j$.

4. The algorithm then lets $\arg\max_j score_j$ be equal to j.

Input:

ACGTTTTACCATATACACAGGCACACACAAACCCAAAACATATTATAATAC
ACGAGGTGTGAGTGAGAGATGGAGACACCGGATTGAGAAGCCCCAAC
CACGGGCTGTGAGTGAGAGATGGAGATTTTTAGTTGATATAATATACAAC
ACGTCTCTCATATACACACACACACACCACAGCAACAAGTGCTTATAATA

Output of initial step:

ACG**TTTTAC**CATATACACAGGCACACACAAACCCAAAACATATTATAATAC
ACGAGGTGTGAG**TGAGAG**ATGGAGACACCGGATTGAGAAGCCCCAAC
CACGGGCTG**TGAGTG**AGAGATGGAGATTTTTAGTTGATATAATATACAAC
ACGTCTCTCATATACACACACACACACCACAGCAACA**AGTGCT**TATAATA

Figure 2.1: An example showing the initialization step in Gibbs sampling. The initial step of the Gibbs sampling algorithm chooses a subsequence of a given length from the input sequences uniformly at random. The subsequences shown in bold are the selected subsequences from this step.

ACG**TTTTAC**CATATACACAGGCACACACAAACCCAAAACATATTATAATAC
ACGAGGTGTGAG**✘✘✘✘✘✘**ATGGAGACACCGGATTGAGAAGCCCCAAC
CACGGGCTG**TGAGTG**AGAGATGGAGATTTTTAGTTGATATAATATACAAC
ACGTCTCTCATATACACACACACACACCACAGCAACA**AGTGCT**TATAATA

Figure 2.2: Illustration of step 1 of Gibbs sampling. In step 1 one of the subsequences is chosen randomly and the ℓ-length subsequence is deleted.

It follows that at each iteration of the algorithm, one of the motif instances is replaced with an improved instance. When using the Gibbs sampler, a different set of initially selected subsequences will result in a different final set and therefore, to obtain results that are close optimal, the algorithm has to be ran a number of times and the motif instance with the maximum score is returned. There are several algorithms that use Gibbs sampling as a subroutine. For example, AlignACE (Aligns Nucleic Acid Conserved Elements) [95] and BioProspector [72] are motif-recognition programs that employ a Gibbs sampling strategy.

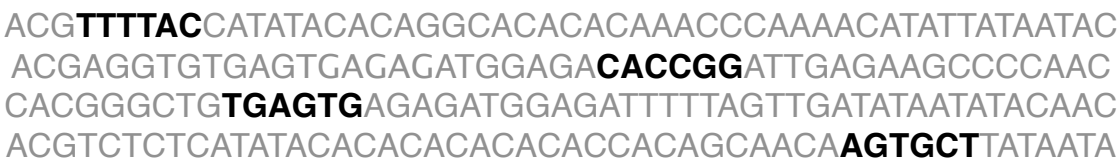

Figure 2.3: Illustration of step 4 of Gibbs sampling. After the score of each ℓ-length in sequence S_i is calculated the subsequence that has the maximum score is added, replacing the deleted subsequence in Step 1.

BioProspector

BioProspector uses Gibbs sampling to search a list of sequences for potential regulatory motifs [72]. Gibbs sampling first initializes the motif matrix using length-ℓ subsequences that are randomly selected from the input; then, it samples from all length-ℓ subsequences in the input sequences to update the PWM. The probability of selecting a length-ℓ subsequence is proportional to the likelihood of generating it from the current motif matrix over the likelihood of generating it from the non-motif background. The motif matrix is updated until convergence, or until a certain number of sampling iterations has been reached. BioProspector has several significant improvements compared to GibbsDNA. It uses a Markov model estimated from all promoter sequences in the genome to model adjacent nucleotide dependency and improve motif specificity. It also adopts two thresholds to allow each input sequence to contain zero or multiple copies of the motif [72].

MDScan

MDScan [73] is a motif-discovery program that combines two widely adopted motif search strategies: word enumeration [25, 101, 111, 114] and position-specific weight matrix updating [6, 59, 72]. It has been shown to detect longer motifs with a greater number of degenerate positions, however, it is unable to detect $(15, 4)$-motifs with adequate accuracy [73].

CONSENSUS

CONSENSUS is a greedy algorithm that is based on a matrix consensus of overrepresented motifs in a set of sequences, coupled with a phylogenetic assessment of the statistical robustness of this motif [60].

The YMF Approach

The Yeast Motif Detection (YMF) approach calculates the z-score for each oligomer occurring in the data set and has been shown to successfully detect motifs in yeast genomes. The success of YMF in detecting transcription factor binding sites in the yeast genome is due to the simplicity of the problem: the short motif length and small number of degenerate positions. Unfortunately, YMF is not suitable for detecting motifs where the length is large and the number of degenerate positions is significant.

2.2.3 Combinatorial Methods

Combinatorial approaches to motif-recognition use the motif model given in Definition 1.2. The aim of these methods is to solve progressively weaker and more computationally challenging problems with superior accuracy and efficiency. For example, WINNOWER, which was developed in 2000, was capable of solving of solving the $(15, 4)$-motif problem with 20 sequences, each of which had length at most 700 bp [88], whereas PSM1, which was developed in 2005, was able to solve the $(18, 6)$-motif problem with 20 sequences, each of which has length at most 1000 bp, with increased accuracy. We will describe the following three combinatorial approaches to motif recognition in detail: SP-STAR, WINNOWER [88], and PROJECTION [24].

SP-STAR

SP-STAR, developed by Pevzner and Sze [88], does an enumerative search but only over a restricted search space defined by the data instead of the entire space. First, a set of subsequences is chosen as the initial motif, and then the best possible motif is obtained by employing a local improvement heuristic that uses scoring function that evaluates the strength of the motif. Given a set of length-ℓ sequences $\{x_1, x_2, \ldots, x_n\}$, we define $SPscore$ as $\sum_{i,j} d(x_i, x_j)$. We summarize SP-STAR in the following steps:

1. Let X be the set of all length-ℓ subsequences in the n input sequences $\{S_1, \ldots, S_n\}$.
2. For any length-ℓ sequence $x \in X$ chosen uniformly at random, do the following:
   (a) Find the set of n length-ℓ sequences $\{s_1, \ldots, s_n\}$ such that s_i is contained in S_i for all i, and $d(x, s_i)$ is minimized.
   (b) Let x be a sequence that contains the alphabet symbol that occurs most frequently at each position and with ties broken arbitrarily.

(c) Repeat steps 2a and 2b until $SPscore$ of $\{s_1, \ldots, s_n\}$ cannot be further improved.

The algorithm is rerun a number of times and the set of sequences that minimizes $SPscore$ is returned. Further, the authors suggest that the local improvement steps "may take a long time" [88] and therefore, should only be performed on a fraction of the best initial sets of sequences. Nonetheless, the number of sequences to be searched is approximately $\binom{\ell}{d}3^d$. SP-STAR was successful in finding $(15, 4)$-motif instances in data sets containing 20 sequences, each of which has maximum length 700 but failed to have reasonable accuracy when the sequence length exceeded 700 [92].

WINNOWER

WINNOWER is a simple graph-theoretic approach that represents a motif instance as a clique[1]. WINNOWER first constructs a graph $G(V, E)$ by creating a vertex for every subsequence of length ℓ occurring in the input and adding an edge between all pairs of vertices corresponding to subsequences that are distance $2d$ apart and occurring in different input sequences. Now the problem of finding a (ℓ, d)-motif corresponds to finding a clique of size n. However, clique finding is NP-complete [51] and therefore, WINNOWER proposes a method to detect and delete spurious edges to reveal sets of vertices whose corresponding subsequences are possible motif instances [88]. We note that G is a n-partite graph[2].

We define a vertex to be a *neighbour* of a clique C if it is connected to every vertex in C, and a clique to be *extendable* if it has at least one neighbour in each of the n partitions of G. The algorithm is based on the observation that every edge in a maximal n-clique belongs to at least $\binom{n-2}{k-2}$ extendable cliques of size k. An edge is referred to as *spurious* if it does not belong to any extendable clique of size k. WINNOWER then iteratively deletes spurious edges for increasing values of k with the idea that only extendable cliques remain.

When $k = 1$, a vertex u is a neighbour of a vertex v if $(u, v) \in E$. Every vertex that does not contain at least $n - 1$ neighbours is deleted (since it certainly cannot be in a clique of size n). For $k = 2$, a vertex u is a neighbour of an edge v, w if v, w, and u form a triangle. Therefore, any edge that does not have $n - 2$ neighbours that are from different partitions is deleted from G. For $k > 2$, WINNOWER relies on the observation that for any clique of size n, there are $\binom{n}{k}$ extendable cliques with k vertices and thus, every edge on a n-clique must belong to at least $\binom{n-2}{k-2}$ extendable cliques of size k. Any edge that is not contained in the required number of extendable cliques is deleted.

[1] A clique is a graph in which every pair of vertices is joined by an edge.

[2] A n-partite graph is a graph where the vertices can be partitioned into n disjoint sets and there are no edges between vertices in the same set.

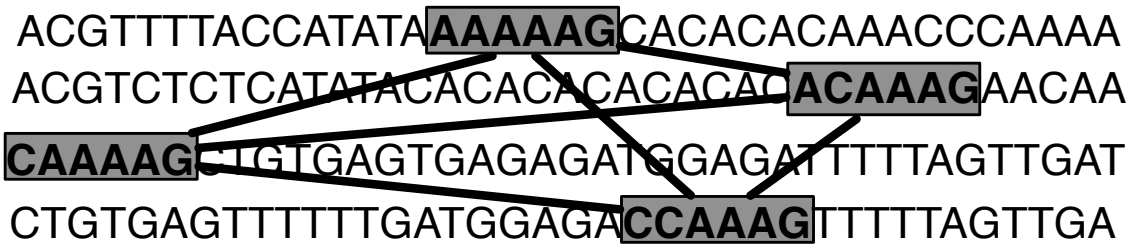

Figure 2.4: An example showing a clique, which does not correspond to a motif set, in the graphical representation of the input used by WINNOWER. Let $\ell = 6$, $d = 2$, and $n = 4$. In this example, we have set of sequences where each pair of sequences has distance at most $2d$ but there does not exist a length-ℓ sequence that has distance at most d from each of the sequences in the set.

One of the pitfalls of WINNOWER is that it does not actually report any motifs; it is only concerned with identifying cliques containing n vertices. Not all cliques correspond to valid motifs (according to the definition of Pevzner and Sze [88]) and therefore, further analysis is required to distinguish cliques corresponding to valid motifs from those that are not. See Figure 2.4 for an example of a set of sequences that corresponds to a clique but not a motif.

The running time of the algorithm is $O(N^{2d+1})$, where $N = nm$, and m is length of the input sequences. The algorithm performs better than SP-STAR with the same data sets. However, due to the large number of spurious edges, the running time is prohibitively large and grows very rapidly as the motif strength weakens or subsequence length or number increases [71].

There have been many motif-recognition programs that have extended WINNOWER since its initial development. In 2003, Liang *et al.* [71] presented cWINNOWER, which is essentially the WINNOWER algorithm with added constraints on the graph construction whose purpose is to reduce the number of spurious edges in the initial graph. Sze *et al.* [107] extended the graph formation of WINNOWER [88]; they formulate the motif-finding problem as the problem of finding large cliques in k-partite graphs, with the additional requirement that there exists a string s that is close to every motif instance. Yang and Rajapakse [118] adopted a graph formulation similar to Pevzner and Sze [88], then use a dynamic-programming approach to search for each clique of size n. The motif instances and center strings are then derived during the clique finding. Finally, a rescanning of the data set is done with the center string or the motif in order to find motif instances with Hamming distance less than or equal to d from the center string. Lastly, PRUNER [98]

again uses the same graph formulation as Pevzner and Sze [88] but deciphers between the cliques corresponding to a motif by further restricting the criteria for deleting an edge. All of these algorithms – cWINNOWER, the work of Sze *et al.* [107], the work of Yang and Rajapakse [118], and PRUNER – suffer from the limitation that an exorbitant amount of time and space is needed to obtain decent results [71, 98, 107, 118].

Random Projection

In 2002, Buhler and Tompa developed PROJECTION, an algorithm based on random projections [24]. Assume that we have an (ℓ, d)-motif problem, and s_m denotes the motif that we are interested in detecting. PROJECTION considers all length-ℓ subsequences in the input set, and projects each of these subsequences along k randomly chosen positions, where k is some appropriately chosen value. Hence, for every subsequence in S, generate a k-mer u' that is a subsequence of u corresponding to the k random positions chosen. Note that the random positions are the same for all the length-ℓ subsequences. Hence, each length-k subsequence is thought of as an integer. We group the length-k subsequences according to their integer values (*i.e.* all the length-ℓ subsequences are hashed using the length-k subsequences of any length-ℓ subsequence as its hash value).

If a hashed group has at least a threshold number α of length-ℓ subsequence in it then there is a good chance that s_m will have its length-k subsequence equal to the length-k subsequence of this group. Hence, choosing appropriate parameters is vital to the accuracy and efficiency of PROJECTION. There are $n(m-\ell+1)$ length-ℓ subsequences in the input and there are 4^k possible length-k subsequences. Therefore, the expected number of length-ℓ subsequences that hash into the same bucket is $n(m-\ell+1)/4^k$. The threshold value α is chosen to be twice this expected value. The value of k has to be chosen small enough so that a significant number of motif instances are grouped together under the projection but large enough so that non-motif instances are not grouped together with motif instances; *i.e.* k is chosen such that $n(m-\ell+1) < 4^k$ so that the expected number of random length-ℓ subsequences that hashed into the same bucket is less than one.

The process of random hashing is repeated r times to ensure that that a bucket size of at least α is observed at least more than once. The value of r is defined with respect to several other values which we now define. The probability p that a given motif instance hashes to the planted bucket is equal to $\binom{\ell-d}{k}\binom{\ell}{k}$, and it follows that the probability that fewer than α hash into a bucket is $p' = \sum_{i=1}^{\alpha-1}\binom{n}{i}p^i(1-p)^{\alpha-i}$. Thus, the probability that fewer than α hash into a bucket in each of the r trials is equal to p'^t, which we denote as P. The value of r is equal to $\lceil \frac{\log P}{\log p'} \rceil$. In the original paper of Buhler and Tompa, a value of 0.05 is used for P [24]. In the final step of the algorithm, the sets of length-ℓ subsequences that pass the threshold are collected and processed to obtain the motif sequences that are returned.

The Voting Algorithm

In 2005, Leung and Chin [30] described two algorithms: the Voting algorithm and the Voting algorithm with projection. The latter is capable of solving challenging motif problems (*i.e.* the $(9, 2)$, $(11, 3)$ and $(15, 5)$ problems), as well as detecting motifs where ℓ and d are "large" (*i.e.* the $(20, 7)$ and $(30, 11)$ problems). However, the Voting algorithm (by itself) cannot solve any motif problem when $\ell > 15$ due to heavy time and space requirements [30]. The Voting algorithm with projection uses a similar algorithm as a preprocessing step, then the Voting algorithm is used to recover the motif instances from the set of sequences hashed into the same bucket by projection.

We give a brief description of the Voting algorithm. We define a length-ℓ sequence s' to be a *d-variant* of another length-ℓ sequence s if the Hamming distance between s' and s is at most d. We denote all the d-variants of a sequence s as $N(s, d)$. Note that if s_m is a motif then s_m, along with all the instances are contained in $N(s_m, d)$. Then for each length-ℓ subsequence s_i in the input sequences gives a vote to each of the sequences in $N(s_i, d)$. Further, each length-ℓ sequence can get at most one vote from each of the n input sequences, which ensures that a motif s_m gets exactly n votes. Although the basic Voting algorithm performs better than the basic enumeration algorithm, the required amount of space grows exponentially with d, rendering it impractical for reasonable sized values of ℓ and d.

Pattern Branching

Pattern Branching is a local search algorithm that begins from subsequences that appear in the set of input sequences and systematically alters these initial sequences by changing one letter at a time in order to search for the center sequence [94]. If s' is any length-ℓ sequence then there are $\binom{\ell}{d}3^d$ length-ℓ sequences that have Hamming distance d from s'. We refer to this set of sequences at the *neighbourhood* of s'. One solution to the (ℓ, d)-motif problem is to consider the neighbourhood of each of the ℓ-mers in the set of input sequences, score each of the sequences appropriately, and then return the sequence with the highest score. Since there could be at most $\binom{\ell}{d}3^d(n(m - \ell + 1))$ sequences, the space and time requirements of this basic algorithm are prohibitively large. Nonetheless, there are several algorithms that use this approach [50, 116].

Rather than considering all length-ℓ sequences in the neighbourhood of all occurring subsequences in S, Pattern Branching only examines a selected subset of neighbours. Again, we denote $N(s, d)$ to be all d-variants of s. For any input sequence S_j let $d_{min}(s, S_j)$ denote the minimum Hamming distance between s and any length-ℓ subsequence of S_j. Lastly, denote $d_{min}(s, S)$ to be $\sum_{j=1}^{n} d_{min}(s, S_j)$. For any ℓ-mer s_{ij} in the input sequence S_j let $BestNeighbour(s)$ be equal to the sequence in $N(s_{ij}, d)$ whose distance $d_min(s_{ij}, S)$ is

minimum from all other sequences in $N(s_{ij}, d)$. Pattern Branching starts from a sequence s_{ij}, identifies $s_{ij}^1 = BestNeighbour(s_{ij})$, then identifies $s_{ij}^{k+1} = BestNeighbour(s_{ij}^k)$ for all $1 \leq k \leq d-1$. The best s_{ij}^d among all possible length-ℓ subsequences of the input S is output.

Profile Branching

Profile Branching [90] is similar to Pattern Branching; the main difference between the two algorithms is that in the latter, the search space is of *motif profiles*, rather than motif sequences. Given an length-ℓ sequence $s = s(1), s(2), \ldots, s(n)$, we define the *profile* of s to be a $4 \times \ell$ matrix $X(s)$ where each column is defined as follows:

$$x_{i\alpha} = \begin{cases} \frac{1}{2} & \text{if } s(i) = \alpha, \\ \frac{1}{6} & \text{otherwise} \end{cases}$$

For example, suppose $s = ACCA$ then the motif profile for s is:

	1	2	3	4
A	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{2}$
C	$\frac{1}{6}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{6}$
G	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$
T	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

The algorithm proceeds identically to Pattern Branching but now the sequences in are ranked according to a new score function, which is referred to as the *entropy score*. The Pattern Branching algorithm outperforms the Profile Branching algorithm; however, the latter does not obtain acceptable accuracy when the degeneracy d is large relative to the sequence length ℓ [90].

2.2.4 Challenge Problems

Pevzner and Sze defined the weak motif-recognition problem more concretely by illustrating the limitations of most motif-recognition programs; their results illustrate that although most methods at the time were capable of finding motifs of length 6 with no degeneration, most failed to detect motif instances of length 15 with 4 degenerate positions in a random sample containing 20 sequences, each of which has length 600 bp [88]. Since this problem was defined, many approaches have been developed with the intention of detecting motifs that have a relatively large number of degenerate positions. Pevzner and Sze coined the

phrase "challenge problems" to denote problems where the motif degeneracy (*i.e.* parameter d) is relatively large with respect to the motif length (*i.e.* the parameter ℓ). Since then the phrase has been commonly used and pertains to problems such as the $(12, 3)$, $(15, 4)$, and $(18, 6)$ motif problems as well as many others.

2.2.5 Exact Verses Approximate Algorithms

Algorithms that are exact are referred to as *exhaustive enumeration algorithms* in the literature. Many exact motif-recognition algorithms are known (*e.g.* [21, 50, 81, 92, 100, 104, 108, 112], however, the running times of these algorithms are prohibitively large for any of the challenge problems [24]. Exact algorithms are guaranteed to determine the motif and motif instances, whereas, the solution determined by an approximate algorithm is likely to mismatch with the correct motif and motif instances at a number of positions. We define the *success rate* of a given program using the performance coefficient used by Pevzner and Sze [88], Buhler and Tompa [24], and others [30, 31].

Definition (**performance coefficient**) Let K denote the set of $t\ell$ base positions in the t occurrences of the planted motif, and let P denote the corresponding set of base positions in the t occurrences predicted by an algorithm. The algorithm's success rate is defined as $|K \cap P|/|K \cup P|$.

A performance coefficient of 0.75 or greater is acceptable for algorithms not guaranteeing exact accuracy. The success rate of a program will arise when evaluating motif recognition programs.

Approximate algorithms, such as CONSENSUS [60], Gibbs [68], MEME [6], and ProfileBranching [90] take significantly less time but perform poorly on finding weak motifs; for example (15,4)-motifs are almost completely undetectable using these programs [59]. More recently, there have been a number of programs that are approximate but have decent performance coefficients (over 75%), including PROJECTION [24], the Voting algorithm [30], PSM1 [92] and PMSprune [35]. Unfortunately, most programs failed to achieve decent performance coefficients in practical running time for all (ℓ, d)-motifs we tested. The details of these experiments will be discussed in depth in later chapters.

2.3 Past Work on the String Selection Problems

The Closest String problem has not only generated a significant amount of research interest in itself, but is also an important subproblem of motif recognition. We will discuss some past results concerning this problem, and later in this thesis, we develop and apply some efficient heuristics for this problem.

2.3.1 The Closest String Problem

Stojanovic *et al.* [105] provided a linear-time algorithm for the case $d = 1$. There exists a trivial fixed parameter tractability algorithm when ℓ remains fixed: simply try all possible $|\Sigma^\ell|$ strings. Gramm *et al.* [57] demonstrated that the Closest String problem is in FPT when the number of strings n remains fixed. This parameterized tractability result is based on an integer linear programming formulation with a constant number of variables, and the application of the result of Lenstra [69], which states that integer linear programming is polynomial-time solvable when the number of variables remains fixed. Unfortunately, such an integer linear programming formulation is only of theoretical interest since the corresponding algorithms lead to very large running times even when the number of strings is small. When d is fixed, the problem can be solved in $O(nl + nd(d+1)^d)$ time [57]. Ma and Sun gave an $O(n|\Sigma|^{O(d)})$ algorithm, which is a polynomial-time algorithm when $d = O(\log n)$ and Σ has constant size [77]. Most-recently, Wang and Zhu designed an $O(\ell n + nd|\Sigma|^d 2^{3.25d})$ time fixed parameter algorithm for the Closest String problem [115]. Chen *et al.* [28] and Zhao and Zhang [119] also improved upon the fixed parameter tractable result of Ma and Sun [77].

Ben-Dor *et al.* [10] used a standard linear programming and randomized rounding technique to develop a near-optimal algorithm for large d (*i.e.* when d is super-logarithmic in the number of strings). Lanctot *et al.* [66] gave an approximation algorithm for the Closest String problem that achieves a worst case performance ratio of 2. It is a simple algorithm that constructs an approximate feasible solution in a pure random fashion. Starting from an empty solution, the algorithm selects at random the next element to be added to the solution under construction. In subsequent work, Lanctot *et al.* [67] gave a polynomial-time algorithm that achieves a $\frac{4}{3} + o(1)$ approximation guarantee. Independently, Gąsieniec *et al.* [52] gave a $\frac{4}{3}$-approximation algorithm that uses a similar technique. This improved algorithm is based on a linear programming relaxation of an integer programming model of the Closest String problem. The basic idea consists of solving the linear programming relaxation[3] and using the result of the relaxed problem to define an approximate solution to the original problem. More specifically, Lanctot *et al.* [67] used randomized rounding for obtaining an integer 0-1 solution from the continuous solution of the relaxed problem; this technique works by letting a Boolean variable x in the linear program be equal to one with probability y, where y is the value of the continuous variable corresponding to x in the relaxation of the original integer program.

Using randomized rounding again, Li *et al.* [70] proved the existence of a PTAS for this problem. Unfortunately, the high degree in the polynomial complexity of the PTAS

[3]The linear programming relaxation of an integer program is the problem that arises by replacing the constraint that each variable must be an integer in the set $\{0, 1, \ldots, k\}$ by a weaker constraint, that each variable belong to the interval $[0, k]$.

algorithm renders this result only of theoretical interest. For a given value of r, all choices of r subsequences of length l are considered from the n sequences. The algorithm runs in $O(l(nm)^{r+1})$-time, which is polynomial for any constant r. In 2005, Brejová *et al.* [22] proved the existence of sharper upper and lower bounds for the PTAS on a slight variant of the Closest String problem (which they refer to as the Consensus Pattern problem) and in 2006, Brejová *et al.* [23] improved upon the analysis of the PTAS for various random binary motif models by showing that there are known "weak" instances for which the approximation ratio is $1 + \Theta(1/\sqrt{r})$ [22], and "strong" instances for which the PTAS will be guaranteed to determine the correct answer in efficient time. Andoni *et al.* [4] gave a novel PTAS that has improved time complexity, and most recently, Ma and Sun [77] presented a PTAS with time complexity $O(n^{\Theta(n^{-2})})$, which is currently the best known.

2.3.2 The Farthest String Problem

Lanctot *et al.* proved the existence of a PTAS for this problem, though the high degree in the polynomial complexity of the PTAS algorithm renders this result only of theoretical interest [67]. The algorithm is based on the randomized rounding of the relaxed solution of an integer programming formulation. There have been no improvements on the running time of this PTAS.

In 2004, Cheng *et al.* [29] gave a $O((|\Sigma|(\ell-d_f))(\ell-d_f))$ fixed parameter algorithm for the Farthest String problem. This fixed parameter tractability algorithm is nearly identical to the algorithm of Gramm *et al.* [57] for the Closest String problem. Most recently, Wang and Zhu gave a $O(\ell n + nd2^{3.25d_f})$ fixed parameter algorithm for the Farthest String problem when the input strings are binary strings [115].

2.4 Summary

In this chapter, we surveyed some of the important results in motif recognition, the Closest String problem, and its variants. We described two motif-recognition models, including the combinatorial model that was formally introduced by Pevzner and Sze [88]. Since the combinatorial version of this problem was initially described, there has been a plethora of research in developing motif-recognition programs that are efficient and capable of solving computational challenging instances of this problem. We continue this area of research by developing an efficient motif-recognition program that is based on a novel weighted graph model, and illustrating how combinatorial and probabilistic insights can be applied to gain even more efficiency and capability in solving computationally challenging instances of motif recognition.

Chapter 3

MCL-WMR: a Comprehensive Motif-Recognition Algorithm

In Chapter 2 we described the motif-recognition problem in detail, and described several other motif-recognition programs. Pevzner and Sze defined the weak motif-recognition problem more concretely by illustrating the limitations of most motif-recognition programs; their results illustrate that although most methods at the time were capable of finding motifs of length 6 with no degeneration, most failed to detect motif instances of length 15 with 4 degenerate positions in a random sample containing 20 sequences that were 600 nucleotides long [88]. Since these challenge problems were defined, many approaches were developed with the intention of detecting motifs that have a relatively large number of degenerate positions. In this chapter we describe a new approach for this problem, and provide theoretical and experimental results that support our approach.

Our algorithm, MCL-WMR, builds a weighted graph model of the input data and uses a graph clustering algorithm to quickly determine important subgraphs that need to be searched further for valid motifs. We compare the results of MCL-WMR to that of previously developed approaches; testing on synthetic data has shown that MCL-WMR has competitive running time capabilities and accuracy. An added advantage of MCL-WMR is the ability to detect multiple motif instances.

As previous described, the Closest String problem asks, given a parameter d and a set $S = \{s_1, \ldots, s_n\}$ of n strings, each of length ℓ, whether there exists a string that has Hamming distance at most d from each of the strings in S. The following definition will be useful in this and later chapters:

Definition (weight of a set of sequences) Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of n sequences, each of length ℓ, we define the *weight* of the set S to be $\sum_{i=1}^{n} \sum_{j=i}^{n} \ell - d(s_i, s_j)$.

For a given parameter d we say S is a *motif set* if there exists a center string s^* at distance at most d from any string in S; we say a set S of strings is *pairwise bounded* if the distance between any pair of strings in S is at most $2d$. Every motif set is pairwise bounded; if a pairwise bounded set is not a motif set we say it is a *decoy set*. For example, for $d = 1$ the set $\{000, 001, 010, 100\}$ is a motif set because 000 is a center string for this set. In contrast, the set $\{000, 011, 101, 110\}$ is a decoy set because it is pairwise bounded (since any two of the strings are at Hamming distance at most two) but no center string exists. We introduce the exploration of using the weight of a set as an indicator of whether a Closest String instance is a motif set or a decoy set.

3.1 System and Methods

MCL-WMR involves three stages: graph construction, clique finding using graph clustering, and recovering the motif instances and their center strings. An overview of MCL-WMR is shown as follows:

Algorithm 2 An overview of MCL-WMR

```
% build the graph from the given data
for each of the subgraphs do
    cluster subgraph using MCL
    for each cluster do
        if the weight of the current cluster is above threshold then
            search that cluster to see if it contains a motif
            if we find a motif, we are done.
```

The MCL-WMR algorithm first chooses a *reference sequence* from the data set, denoted as S_r, building the entire graph from all the input data (including S_r), and for each vertex $v_{r\Sigma}$ representing the length-ℓ subsequence from S_r starting at position Σ. We then use the Markov Cluster Algorithm (MCL) [110] to generate subgraphs which contain vertices that are highly inter-related. From these clusters of vertices we will generate the positions of the possible motif instances and their corresponding center string. The algorithm terminates when a motif is found. In order to increase the probability that a motif is found, we minimize searching subgraphs with low probability of containing a motif; hence, the adjacency subgraphs are not clustered and searched in a sequential manner.

3.1.1 Graph Construction

In our graphical representation of the data set, each subsequence of length ℓ is represented by a vertex and the construction of our graph ensures that the motif instances represented

by vertices in the graph are connected to each other and form a clique of size n (though the converse need not hold). Thus, the weak motif-recognition problem is converted to finding cliques with size n in our constructed graph G which is defined as follows:

1. The vertex set contains a vertex $v_{i,j}$ representing the ℓ-length subsequence in sequence i starting at position j, for each i and $j = 1, 2, \ldots, m - \ell + 1$. There are $n(m - \ell + 1)$ vertices.

2. Each pair of vertices $v_{i,j}$ and $v_{i',j'}$, for $i \neq i'$ is joined by an edge when the Hamming distance between the two represented subsequences is at most $2d$.

3. An edge between vertices having distance k has weight $\ell - k$ for $d < k \leq 2d$, or $10(\ell - k)$ for $k \leq d$. This emphasizes subsequences at small distances.

This graph is represented by a symmetric adjacency matrix, where each entry is 0 for a non-edge, or a positive weight for an edge. This representation takes $O((n(m - \ell + 1)^2))$ space and is constructed in $O(n^2(m-\ell)(m+\ell))$ time. Since the graph is n-partite, a clique of size n contains exactly one vertex from each sequence.

We reduce the size of the instance being passed to MCL by considering subgraphs $\{G_0, G_1, \ldots, G_{m-1}\}$, where G_i is the subgraph induced by the vertex corresponding to the reference sequence (denoted as $v_{R,i}$), and its neighbours (for some arbitrary choice of reference sequence R) instead of searching all of G at once. Therefore, we have reduced the problem of finding weak motifs in the input data set S to finding cliques of size n in the G_i, though, some cliques may not correspond to valid motifs. Figure 3.1 illustrates this weighted graph construction of the input sequences.

3.1.2 Using Clustering to Find Motifs

A *clustering* of graph is a decomposition of the vertex set into subsets of vertices that are highly intra-connected and, hence, induce dense subgraphs. A *good clustering* of a graph is an approximation of a partitioning of the graph into cliques. A clique corresponding to a motif will exist in one of the subgraphs of G since each motif instance appears as a vertex in a clique of size n. We use MCL [110] to cluster the sets of vertices to determine subgraphs that are highly intra-connected with high-weight edges, and sparsely inter-connected and thus likely to correspond to a motif instance. We chose MCL because it is able to handle weighted graphs, and because it efficiently handles large undirected graphs. The idea underlying the MCL algorithm is that dense subgraphs correspond to regions where the number of length k paths is relatively large, for small k. Random walks of length k have higher probability for paths beginning and ending in the same dense region than for other paths.

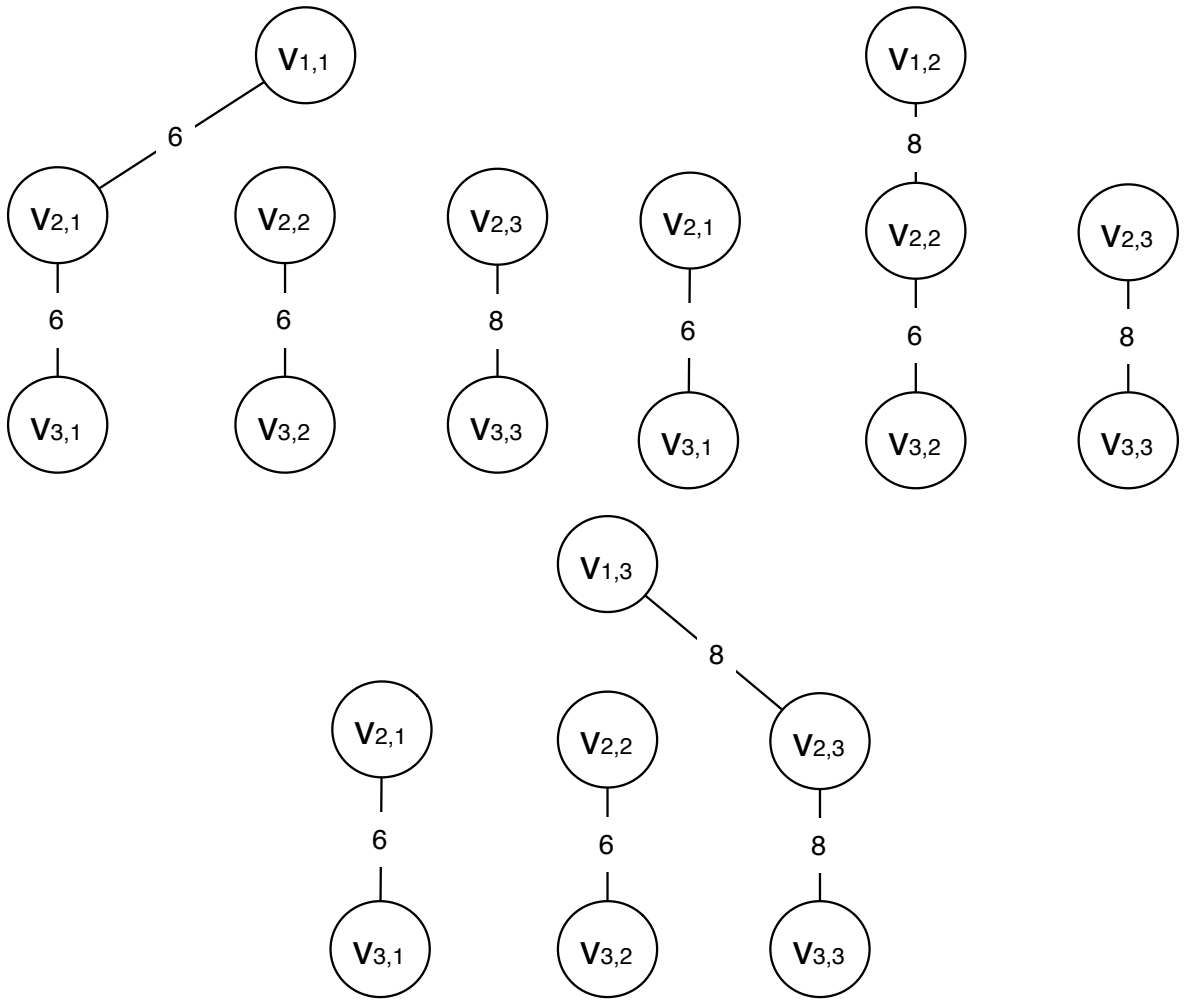

Figure 3.1: An example showing the weighted graph representation of the input data used by MCL-WMR. The three input sequences are shown at the bottom-left corner of the figure. We have $n = 3$, $m = 8$, $\ell = 6$, $d = 1$.

The MCL process consists of the alternation of two operations, namely *expansion* and *inflation*. The image of a stochastic matrix under expansion is just some finite power of the matrix. The image of a (column) stochastic matrix under inflation is formed by raising each entry of the matrix to the same positive power, followed by a rescaling of the columns such that the result is stochastic again. An MCL process is defined by a column stochastic input matrix T_1 and two rows of exponents $e_{(i)}$, $r_{(i)}$, resulting in a row of stochastic matrices $M_{(i)}$, defined by:

$$T_{2i} = \exp_{e_i}(T_{2i-1}),\ i = 1, \ldots, n$$

$$T_{2i+1} = \Sigma_{r_i}(T_{2i}),\ i = 1, \ldots, n$$

where $\exp_{e_i}(T)$ is T^{e_i} and $e_i \in \mathcal{N}$ and $r_i \in \Re$, $r > 0$. The image under the *inflation operator* Σ_r of a nonnegative matrix $M \in \Re^{k \times \ell}$ having no zero columns is defined as:

$$(\Sigma_r M)_{pq} = (M_{pq})^r \sum_{i=1}^{k} (M_{iq})^r.$$

The largest transition probabilities will generally correspond with nodes lying in the same natural cluster; this effect is boosted in the MCL process via the inflation operator. By expansion, transition probabilities corresponding with random walks of higher length are computed. The inflation operator promotes larger transition probabilities at the cost of small probabilities and thus, promotes random walks between nodes lying in the same cluster and demotes random walks between nodes lying in different clusters.

3.1.3 Recovering Motifs

MCL produces sets of vertices that produce dense regions of the subgraph G_i; we filter the subgraphs obtained from MCL to subgraphs that have high probability of containing a motif. A clique in G that represents a motif instance must have size n and have weight greater than or equal to $(\ell - 2d)\binom{n}{2}$ since each pair of vertices are adjacent. We filter out clusters that do not meet these criteria. Clusters that pass this test may contain multiple cliques formed by choosing different subsets of n cluster vertices, or possibly no cliques at all. We identify all ways of forming a clique from the cluster vertices by using the n-partite nature of the graph to explore all possible cliques with the following depth-first search algorithm. The dynamic-programming algorithm we use is as follows:

1. Let the sets $\mathcal{C} = \mathcal{C}_r, \mathcal{C}_2, \ldots, \mathcal{C}_n$ represent the subsets of vertices in a cluster that need to be considered. Hence, $\mathcal{C}_r = \{v_{r\sigma}\}$, $\mathcal{C}_2 = \{v_{21'}, v_{22'}, \ldots, v_{2|\mathcal{C}_2|'}\}$, …, $\mathcal{C}_n = \{v_{n1'}, v_{n2'}, \ldots, v_{n|\mathcal{C}_n|'}\}$. Note that $\mathcal{C}_r$ contains only one vertex since it is the reference sequence.

2. Lists $\mathcal{L}_{ij'}$, for $i = r$ and $j' = \sigma$ and $i = 2, \ldots, n$ and $j' = 1, \ldots, |\mathcal{C}_i|$ are created as follows:

   (a) Set $\mathcal{L}_{r\sigma} = \emptyset$.

   (b) Set $\mathcal{L}_{2j'} = \{v_{r\sigma}\}$ for vertex $v_{2j'} \in \mathcal{C}_2$.

   (c) for $i = 3, \ldots, n$

   Set $L_{ij'} = \emptyset$ for $v_{ij'} \in \mathcal{S}_i$

   for each vertex $v_{i-1k'} \in \mathcal{C}_{i1}$, $k' = 1, \ldots, |\mathcal{C}_{i1}|$

   - if $d(v_{ij'}, v_{i-1k'}) \leq 2d$ then let $\mathcal{L}_{ij'}$ be equal to $\mathcal{L}_{ij'} \cup \{v_{i-1k}\} \cup \mathcal{L}_{i-1k'}$

3. If a clique with size n exists then there must exist a list $\mathcal{L}_{nj'}$ for vertices $v_{nj'} \in \mathcal{C}_n$ that contains vertices from each set $\mathcal{C}_i$ for $i = 1, \ldots, n-1$.

For each such clique found using the above dynamic-programming algorithm, we test if it represents a motif instance by attempting to find a string that has distance at most d to every vertex. We do this by building up a list of possible center strings and the number of mismatches to each vertex for each possibility, one character at a time. Once a candidate string has $d+1$ mismatches to some vertex, it is discarded. Although the space of 4^ℓ possible center strings is very large, in practice the list is pruned very rapidly on the $d+1$st character, *i.e.* after reaching size 4^d.

We use a heuristic to determine a center string which tries all possible length ℓ DNA sequences that have high probability of having distance at most d from each string in the set S and thus, determines whether the subsequences are (ℓ, d)-motifs. In practice this dynamic-programming algorithm is quite efficient because the size of the clusters returned by MCL is small. After obtaining the possible motif instances from s_n, the corresponding likely center string is found by alignment of the n sequences and lastly, in order to determine if the set of subsequences found are motif instances it is checked that each subsequence is of distance at most d from the center string. As the sequences become longer or d becomes large relative to ℓ, the number of spurious cliques found will increase; this final step guarantees that the found subsequences are proper motif instances.

3.2 Analysis of Graph-Theoretic Model

To validate our approach of using edge weights to assist in the search of valid motifs we show the existence of a separation between the weight of a set of sequences corresponding to a motif set and that of a set of sequences that does not correspond to a motif set. We prove that the weight of an arbitrary set of subsequences corresponding to a motif set deviates from the mean with low probability. Empirical results support the analytical

description of the distribution and show that for a typical motif-recognition problem (*i.e.* when $\ell = 15$ and $d = 4$) there exists a separation between the distribution of the weights of the motif sets and those of the decoy sets.

3.2.1 Analysis of the Weight of a Clique Containing a Motif

Consider a clique C that contains a set of subsequences corresponding to a motif set. Let W be the random variable for the sum of each of the $\binom{n}{2}$ edge weights in C. Let $v_1, v_2, \ldots v_n$ be the set of n vertices in C corresponding to sequences $s_1, \ldots, s_n$. We seek the mean of W and a tail bound for large deviations from the mean. Let W_i be the expected value of the random variable W given that the first i subsequences in C are known.

We prove several results concerning random (ℓ, d)-motif sets. A standard method used to generate a random (ℓ, d)-motif set is to choose an ℓ-length sequence uniformly at random from all possible $|\Sigma|^\ell$ sequences to be the center string, and then form a motif set by selecting n sequences at random with replacement from the set of all sequences with Hamming distance at most d from this center string [24]. This samples motif sets with probability proportional to the number of distinct center strings it has and thus, corresponds to how synthetic problem data sets are constructed and how we expect meaningful motif sets arise in nature. For example, synthetic problem instances are traditionally generated as follows: a random center string of length ℓ is chosen, n occurrences of the motif are generated by randomly mutating at most d positions, and each of the n motif instances is embedded at a random location into a different background string of length m. We note that other non-uniform distributions have also been used to generate motif sets [88].

Theorem 3.2.1 *The expected weight of a clique in G, which models a random (l, d)-motif problem containing n sequences, is*

$$E[W] = \binom{n}{2}\left(\ell - \frac{1}{\beta^2}\sum_{a=0}^{d}\sum_{b=0}^{d}\binom{\ell}{b}\binom{\ell}{a}3^{a+b}\left(a+b-\frac{4ba}{3l}\right)\right)$$

where $\beta = \sum_{i=0}^{d}\binom{\ell}{i}3^i$.

Proof Given an (ℓ, d) motif set S, we aim to compute the expected value of the weight of the set S (*i.e.* $E[\sum_{i=1}^{n}\sum_{j=i+1}^{n}(\ell - d(s_i, s_j))]$. Let μ_e be $E[d(s_i, s_j)]$ for any pair of sequences s_i and s_j, where the corresponding vertices v_i v_j are joined by an edge in the clique that contains S.

$$E\left[W\right] = E\left[\sum_{\forall v_i, v_j, i<j}(\ell - d(s_i, s_j))\right] = \binom{n}{2}\mu_e$$

We choose the n sequences uniformly from the $\beta = \sum_{i=0}^{d} \binom{\ell}{i} 3^i$ possible choices. Let s be a center string for the α_i denote the Hamming distance between s_i and the center string s. The expected weight of an edge will depend on the distance of the two sequences from the center string, so we break the expectation into pieces, as follows:

$$\begin{aligned}
\mu_e &= \sum_{\alpha_i=0}^{d} \Pr[d(s,s_i)=\alpha_i] E[d(s_i,s_j)|d(s,s_i)=\alpha_i] \\
&= \sum_{\alpha_i=0}^{d} \Pr[d(s,s_i)=\alpha_i] \cdot \sum_{\alpha_j=0}^{d} \Pr[d(s,s_j)=\alpha_j] \\
&\quad E[d(s_i,s_j)|d(s,s_i)=\alpha_i, d(s,s_j)=\alpha_j] \\
&= \sum_{\alpha_i=0}^{d} \frac{\binom{\ell}{\alpha_i}3^{\alpha_i}}{\beta} \sum_{\alpha_j=0}^{d} \frac{\binom{\ell}{\alpha_j}3^{\alpha_j}}{\beta} E[d(s_i,s_j)|d(s,s_i)=\alpha_i, d(s,s_j)=\alpha_j]
\end{aligned}$$

The remaining problem is to compute the expected Hamming distance between s_i and s_j, knowing that the sequences consist of copies of S with a and b positions mutated, respectively. If a position was mutated in neither string it is a match; if a position was mutated in one sequence but not the other it is a mismatch; if a position was mutated in both sequences, it is a match with probability $\frac{1}{3}$.

If s_i is fixed, s_j consists of b mutations that each either hit one of the a mutated positions in s_i or one of the other $\ell - a$ positions, sampled without replacement. The number that hit the a mutated positions in s_i follows a hypergeometric distribution with mean $\frac{ba}{\ell}$. If the number of hits to mutated positions is c, the expected total number of mismatches is: $b - c$ positions that hit among the $\ell - a$ non-mutated positions in s_i, $a - c$ positions among the a mutated positions of s_i that were *not* hit, and $\frac{2}{3}c$ mismatches from the hits among the mutated positions, for a total of $(b-c) + (a-c) + \frac{2}{3}c = a + b - \frac{4}{3}c$. Therefore, $E(d(s_i,s_j)|d(s,s_i)=a, d(s,s_j)=b) = a + b - \frac{4ba}{3\ell}$.

Therefore, μ_e is equal to $\ell - \frac{1}{\beta^2} \sum_{a=0}^{d} \binom{\ell}{a} 3^a \sum_{b=0}^{d} \binom{\ell}{b} 3^b \left(a + b - \frac{4ba}{3\ell}\right)$, the expected weight of a single edge.

$$\begin{aligned}
E[W] &= \binom{n}{2} \left(\ell - \frac{1}{\beta^2} \sum_{\alpha_i=0}^{d} \binom{\ell}{\alpha_i} 3^{\alpha_i} \sum_{\alpha_j=0}^{d} \binom{\ell}{\alpha_j} 3^{\alpha_j} \left(\alpha_i + \alpha_j - \frac{4\alpha_i\alpha_j}{3\ell} \right) \right) \\
&= \binom{n}{2} \left(\ell - \frac{1}{\beta^2} \sum_{\alpha_i=0}^{d} \sum_{\alpha_j=0}^{d} \binom{\ell}{\alpha_j} \binom{\ell}{\alpha_i} 3^{\alpha_i+\alpha_j} \left(\alpha_i + \alpha_j - \frac{4\alpha_i\alpha_j}{3l} \right) \right)
\end{aligned}$$

□

We are able to bound the probability of W deviating from its mean by first demonstrating that $W_0, W_1, \ldots, W_n$ is a martingale sequence and next, applying Azuma's inequality [83] to determine the probability of a specific deviation.

Theorem 3.2.2 *Consider the (ℓ, d)-motif problem containing n sequences. Let W be the sum of the $\binom{n}{2}$ edge weights in an arbitrary clique in G_{motif} that corresponds to a motif set and let μ_W be the expected weight of the motif set, then for any $\lambda > 0$,*

$$\Pr[|W - \mu_W| \geq \lambda] \leq 2 \exp\left(-\frac{\lambda^2}{2d^2(n+1)}\right)$$

Proof Recall that W_i is the expected value of the random variable W given that the first i sequences in C are known and hence, the distances between the center string s and the first i vertices are known. Without loss of generality we choose a center string s and let F_i be the σ-field generated by the random choice of the sequence s_i from the set of all sequences at most distance d from s. It follows that, $W_i = [W|\mathrm{F}_i]$ since W_i denotes the conditional expectation of W knowing the first i subsequences and therefore, $W_0, W_1, \ldots, W_n$ is a martingale sequence [83], with $W_0 = E[W]$ and $W_n = W$.

We now focus on the value $W_i - W_{i-1}$. Let $\Delta_{i,e}$ be the change in the random variable representing the weight of an edge e from knowing the first $i-1$ sequences to knowing the first i sequences. The value of $\Delta_{e,i}$ is non-zero for edges where the sequence corresponding to one of the endpoints of that edge was previously not known and is now known. Each vertex in the clique is adjacent to $n-1$ vertices; $i-1$ of these vertices have corresponding sequences that were known and $n-i$ of these vertices have corresponding sequences that were unknown; all other $\binom{n}{2} - n + 1$ remaining edges in the clique have no change in the expected value. In both cases, the expectation of the weight of the edge can change by at most d. Again, μ_e is the expected value when neither sequences s_i or s_j corresponding to the endpoints of the edge $e = (s_i, s_j)$ are known and therefore, we have the following:

$$|W_i - W_{i-1}| \leq \Delta_{i,e}(n-1) = d(n-1)$$

We have shown that the random variables $W_0, W_1, \ldots, W_n$ form a martingale with $W_0 = E[W]$ and $W_n = W$ and that $|W_i - W_{i-1}| \leq d(n-1)$ and therefore, we can invoke Azuma's inequality [83] to give us the following for any $\lambda > 0$:

$$\Pr\left[|W - \mu_W| \geq \lambda\right] \leq 2 \exp\left(-\frac{\lambda^2}{2\sum_{i=0}^{n} d^2} =\right) = 2 \exp\left(-\frac{\lambda^2}{2d^2(n+1)}\right).$$

□

We can compare this theoretical tail bound with the distribution of the values obtained from MCL-WMR. Figure 3.2 demonstrates that as the value of n increases, the deviation of the weight of cliques corresponding to motifs will occur with less probability, and the weight of the cliques will become more centralized around the mean. Similar experimental

tests were completed to demonstrate the relationship between the weight of spurious cliques when $n = 15$ and when $n = 50$, specifically, we ran MCL-WMR 100 times with $m = 800$, $\ell = 15$, and $d = 4$ and determined cliques that did not correspond to valid motifs. We found no spurious cliques in the data sets when $n = 50$, agreeing with our intuition that very few spurious cliques occur randomly in the data set when n becomes large.

We note our confidence in MCL-WMR being able to detect cliques – both spurious and those corresponding to motifs – this is due to the accuracy in detecting the embedded motifs (see Section 3.3 details concerning these experimental tests). These results also suggest that when n is relatively large we can be more certain that any cliques found correspond to valid motifs (an attribute that will be further explored in this thesis).

3.2.2 Discussion of Complexity

A few interesting observations can be made regarding the complexity of the algorithm and the quality of its solutions. Finding cliques of maximum size in a given input graph is NP-complete and thus, unlikely to be solved in polynomial-time [51]. Further, the results from Chen *et al.* [27] show that unless unlikely consequences occur in parameterized complexity theory, the problem of finding maximum – size cliques cannot be solved in $n^{o(k)}$ time. Thus, of finding cliques of size k is not likely to be computationally feasible for graphs of significant size. The best known algorithm for finding cliques of size k in a given input graph runs in time $O(n^{ck/3})$, where c is the exponent on the time bound for multiplying two integer $n \times n$ matrices; the best known value for is c is 2.38 [84].

The running time for the straightforward algorithm that checks all size k subsets is $O(n^{k+2})$ and is the one to be most likely to be implemented in practice. The running time of the algorithm of Yang and Rajapakse [118], a dynamic programming clique finding algorithm, is $O(n(mA^2 + A^{m-1}p^{2m-5})$, where $A = n\sum_{i=0}^{2d}\binom{\ell}{i}(3/4)^i(1/4)^{\ell-i}$, n is the length of each sequence and m is the number of sequences. This running time reflects the steep computational expense required to find cliques of a given size for an input graph. Similarly, the estimated running time of the WINNOWER algorithm is $O((nD)^4)$, where D is approximately 30 for the challenge problem [88].

The time required by MCL-WMR to find a solution is not affected directly by the length of the motif that is to be discovered, as is true of many other methods. Rather, it is the weakness of the motif – that is, the probability of the pairwise motif similarity occurring randomly – that has the most impact on the complexity of the algorithm; the increased probability of a clique of high weight affects the running time of MCL-WMR since the exponential-time algorithm required to determine in a high cluster or subgraph contains a motif instance.

We can compare the computational complexity of these programs by considering the

required running time of MCL-WMR of the three sequential steps–that is, the computational time required to construct the graph, finds all cliques of size n, and determine the motifs and center string. Other graphical methods of motif finding rely on an enumeration method to find dense subgraphs, including WINNOWER that requires each edge to be checked, and the algorithm of Yang and Rajapakse that uses dynamic programming on the complete graph. MCL-WMR omits this computational intensive work by employing MCL, which runs in time $O(N^3)$, where N is the number of vertices of the input graph [110]. Hence, the most computationally expensive step is the clique-finding function, whose computation time increases with the number of vertices.

3.3 Experimental Results

We tested MCL-WMR on synthetic problem instances generated according to the embedded (ℓ, d)-motif model. We produce problem instances as follows: first we choose a random string of length ℓ, and pick m occurrences of the motif by randomly choosing d positions per occurrence and randomly mutating the base at each. Lastly, we construct m background sequences of length n and insert the generated motifs into a random position in the sequence. For each of the (ℓ, d) combinations, 100 randomly generated sets of input sequences ($n = 20$, $m = 600$) were generated. An entry "-" indicates that the program was unable to solve the specific problem in a reasonable amount of time.

Our program found the exact location of a motif instance every single time and therefore, the performance coefficient is 1. Other programs we tested did not achieve a performance coefficient of 1 and thus, returned approximate solutions to the embedded motif. The computation time of previous programs that find the exact solution becomes unacceptable as the motifs become degraded beyond the $(15, 4)$ problem [106]. For example, in 2004 Styczyski *et al.* [106] solved the $(17, 5)$ problem exactly but took greater than 3 weeks to do so and similarly, solving the $(14, 4)$ took longer than 3 months (when $n = 20$ and $m = 600$). The main advantage to our tool is the ability to solve the extremely difficult challenge problems (see Table 3.1). See Subsection 2.2.4 for description of the term "challenge problem".

The performance coefficient, which are shown for the various programs tested in Tables 3.1, 3.2, and 3.3, describe the fraction of correctly solved instances and is described previously Subsection 2.2.5. The performance coefficient of MCL-WMR is greater than that of the previous algorithms in every line of Table 3.1. MCL-WMR correctly solved planted $(11, 2)$, $(13, 3)$, $(15, 4)$, $(17, 5)$ and $(18, 6)$ on all data sets – in these cases, the planted motif and motif occurrences at least as strong as planted motifs. WINNOWER, PROJECTION, and SP-STAR achieve acceptable performance on the $(11, 2)$, $(13, 3)$ and $(15, 4)$ problem instances when the sequence length is less than or equal to 600 and the number of se-

(ℓ, d)	Gibbs	PROJECTION	SP-STAR	WINNOWER	MCL-WMR
(10, 2)	0.150	0.80	0.56	0.78	1.00
(11, 2)	0.68	0.94	0.84	0.90	1.00
(12, 3)	0.03	0.77	0.33	0.78	1.00
(13, 3)	0.60	0.94	0.92	0.92	1.00
(14, 4)	0.02	0.71	0.02	0.02	1.00
(15, 4)	0.19	0.93	0.73	0.92	1.00
(17, 5)	0.158	0.93	0.69	0.03	1.00
(18, 6)	-	-	-	-	1.00

Table 3.1: Comparison of the performance on a range of (ℓ, d)-motif problems with synthetic data. Data from other algorithms are from Buhler and Tompa [24]. MCL-WMR, GibbsDNA, WINNOWER, and SP-STAR are averaged over eight random instances, while PROJECTION is averaged over 100 random instances. In all these examples, $n = 20$ and $m = 600$.

m	PROJECTION	MCL-WMR
600	2410 / 102	2208 / 34
800	2662 / 121	2802 / 42
1000	2819 / 131	3223 / 89
1200	3404 / 139	3823 / 102
1400	3808 / 154	4038 / 150
1600	4310 / 159	5037 / 167
1800	4861 / 167	4502 / 182
2000	5012 / 182	5001 / 201

Table 3.2: Comparison of the time required by MCL-WMR and PROJECTION to solve the $(15, 4)$-motif problem with 20 sequences and m varying between 600 to 2000. The mean and standard deviation of the running time in CPU seconds is given. The success rate for PROJECTION was between 0.80 and 0.88.

quences is less than or equal to 20. However, all fail on the $(18, 6)$ and $(19, 6)$ problem, and WINNOWER and SP-STAR fail on the $(16, 5)$ and $(17, 5)$ problem instances. The performance of MCL-WMR is best on the more difficult planted $(14, 4)$, $(16, 5)$, $(17, 5)$ and $(18, 6)$ motif problems when compared to results from previous algorithms. WINNOWER and SP-STAR typically failed to find the planted motifs and PROJECTION often failed to have acceptable performance on the more difficult cases of the challenge problem [24] and hence, MCL-WMR's performance substantially exceeded that of previously algorithms.

We evaluated the performance of MCL-WMR on problem instances with longer back-

ground sequences – that is, problems where m varies from values greater than 600. As the length of the sequences increase, the number of randomly occurring ℓ-mers increases; specifically, the increase in m increases the probability of cliques of high-weight occurring. MCL-WMR will recover more computation time since the graph size will increase with the size of m. Our results demonstrate that the computation time is comparable to that of PROJECTION, however, MCL-WMR is significantly more accurate than PROJECTION. See Table 3.2 for an illustration of these results. Considering the $(15, 4)$ motif problem and fixing the number of sequences to be 8, CONSENSUS, GibsDNA, and MEME have an unacceptable performance rate (*i.e.* below 0.7) when the sequence length increases to 300 or 400, the performance of WINNOWER breaks at length 700, and SP-STAR breaks when the length is 800 to 900.

(ℓ, d)	PROJECTION	MCL-WMR
(10, 2)	400 / 38 (0.98)	687 / 32
(11, 2)	670 / 43 (0.98)	1023 / 34
(12, 3)	1020 / 83 (0.87)	1442 / 52
(14, 4)	1521 / 95 (0.88)	1623 / 78
(17, 5)	5242 / 132 (0.78)	3023 / 103
(18, 6)	-	5132 / 120

Table 3.3: Comparson of the running time of MCL-WMR to PROJECTION on various motif challenge problems. In all these examples, $n = 20$ and $m = 600$. The mean and standard deviation of the running time in CPU seconds is given. The success rate for PROJECTION is included in brackets.

3.4 Summary and Open Problems

We proposed an efficient algorithm for motif-recognition, whose specific purpose is to solve instances where there exists a large amount of degeneration with respect to the motif length. We demonstrated promising results on synthetic data. Specifically, we showed promising running time and accuracy for all challenge problems, with most-impressive improvement on the $(14, 4)$, $(17, 5)$ and $(18, 6)$ problems. Previous algorithms lack accuracy and efficiency in solving all challenge problems.

Most importantly, we gave a novel model and framework for solving motif recognition instances that we will consider and study further in this thesis. By changing the graphical model to incorporate edge weights, we can exploit theoretical results demonstrating the existence of a separation between the weights of cliques corresponding to valid motifs and the weights of those that do not, and obtain improved search techniques. Our theoretical

work and empirical data show a large percentage of the cliques corresponding to valid motifs have total weight in a narrow range. This helps us distinguish cliques containing valid motifs and spurious cliques.

The largest area for further improvement of MCL-WMR is the recovery of the motifs. The dynamic-programming algorithm that explores the dense subgraphs to determine which subgraphs correspond to motif sets, and which correspond to decoy sets requires exponential time. Note that this deciphering of subgraphs essentially reduces to solving instances of the Closest String problem. Next, we aim to create, analyze, and employ efficient algorithms for the Closest String problem.

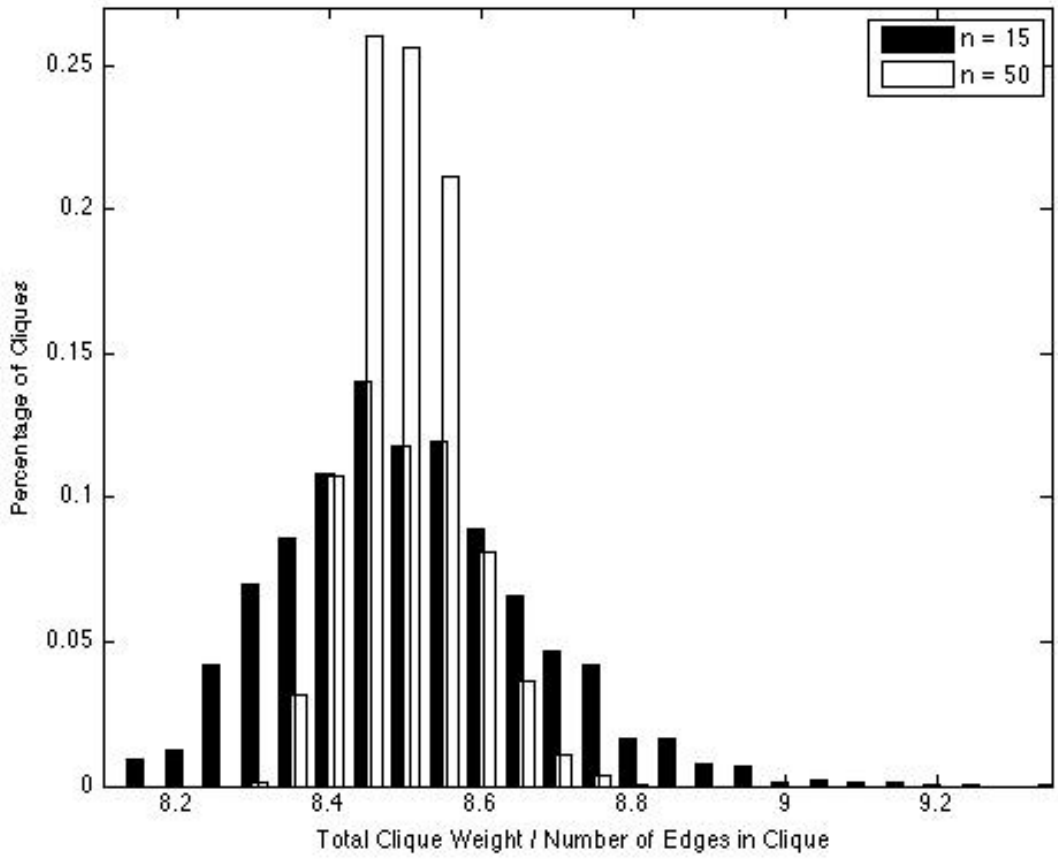

Figure 3.2: Data illustrating the distribution of the mean weight of an edge in a clique that corresponds to a valid motif set of size 15 (shown in black) and 50 (shown in white). The data is given for the $(15, 4)$-motif problem with $m = 600$.

Chapter 4

Algorithms and Analysis for the Closest String Problem

In the previous chapter we described MCL-WMR, a motif-recognition program that builds a weighted graph model of the input data, efficiently clusters the data to determine smaller subproblems that are likely to contain a motif, and recovers motifs by the use of a dynamic-programming algorithm. The last step involves solving the Closest String problem a significantly large number of times. As described in the previous chapter, we refer to a set of strings S as pairwise bounded if for all strings $a, b \in S$, $d(a, b) \leq 2d$. Thus, the Closest String problem reduces to discerning between pairwise bounded sets that have a center string, and if so, finding one such string s^*, and those sets that do not. The dynamic-programming algorithm used by MCL-WMR to solve the Closest String problem has exponential running time and in practice is the most computational intensive step of MCL-WMR. Hence, this algorithm becomes a significant bottleneck in solving motif-recognition instances – especially where d is large relative to ℓ. To improve upon the capability of MCL-WMR, we develop efficient heuristics to solve the Closest String problem.

In this chapter we study the Closest String problem in more detail. Smith showed several results indicating the "alphabet size does significantly influence many pattern finding problems" [102, p. 51]. Similarly, we give strong evidence to demonstrate that the number of strings has a significant affect on the cost of solving the Closest String problem and therefore, influences how we should attempt to solve the problem. We investigate different methods for efficiently discerning between motif sets and decoy sets; to accomplish this task we will divide the problem into studying instances with a small number of strings, and instances with a large number of strings.

Following the path of previous authors [56, 107], we first focus on efficiently solving restricted instances of the Closest String problem. Gramm *et al.* [56] and Sze *et al.* [107] gave direct, combinatorial algorithms for solving the Closest String problem exactly

for three strings. The algorithm of the former authors considers the possible combinations of alphabet symbols that can occur for three strings, then specifies conditions for which a center string can be constructed [56]. Sze *et al.* [107] gave a counting argument to demonstrate a condition for which a set of three strings has a center string and when it does not. Gramm *et al.* stated that the problem of finding an efficient polynomial-time algorithm for solving the Closest String problem on a set of four strings remains open "due to the enormous combinatorial complexity [of the integer linear programming based solution]" [57, p. 13]. We resolve this open problem for binary strings; specifically, we give an exact combinatorial algorithm for four binary strings.

We also prove that Closest String instances can be solved efficiently when the instances have a large number of strings. Evidence for the efficiency is provided in a two-fold manner. First, we introduce and investigate a randomized algorithm for the Closest String problems, which we refer to as *CSP-Greedy*. We prove that this randomized algorithm computes, with high probability, the Closest String on smoothed instances up to a constant factor approximation. Using smoothed analysis, we show the algorithm runs in $O(\ell^3)$, where ℓ is the string length, and achieves a $(1 + f(\epsilon, n, \ell))$ approximation guarantee, where $\epsilon > 0$ is any small value, n is the number of strings, and $f(\epsilon, n, \ell) = O(\delta^n)$ for some $\delta < 1$.

Next, we examine the $O(n\ell + nd \cdot d^d)$-time algorithm by Gramm *et al.* [57], which has been shown to work very efficiently on practical instances. We explain this good performance through smoothed analysis. Essential to our analysis is the introduction of a new perturbation model of the Closest String model and the analysis of the Gramm *et al.* [57] algorithm in this model. We prove for any given Closest String instance I, the average running time of this algorithm on a small perturbation of I is $O(n\ell + nd \cdot d^{2-\epsilon})$.

We refer to the *cardinality* of a Closest String instance as the number of strings in the set. We note that the results in this chapter are complementary; the first part of this chapter will demonstrate that small, very restricted instances can be solved efficiently and the second part will show that instances that contain a large number of strings can be solved efficiently.

We conclude this section by showing the applicability of this analysis to motif recognition. We extend MCL-WMR to incorporate *CSP-Greedy* and obtain *pMCL-WMR*, a program that is able to detect motifs in sets of large cardinality (*i.e.* instances that have 30 or more sequences) and find regulatory sequences in genomic data. Our data shows that detecting motifs in large data sets is easier in comparison to when the number of sequences is relatively small – a fact that gives surprising insight into this central problem in bioinformatics.

The main contributions in this chapter are as follows:

- We give a linear-time algorithm for solving small Closest String instances (Section 4.1).
- We describe our algorithm, *CSP-Greedy*, prove it achieves a 2-approximation guarantee, and give a bound on the probability that *CSP-Greedy* returns an optimal solution to the Closest String instance. Using smoothed analysis we show *CSP-Greedy* achieves an $1 + f(\epsilon, n, \ell)$-approximation guarantee in time $O(\ell^3)$, where $\epsilon > 0$ is any small value and $f(\epsilon, n, \ell)$ approaches zero in time that is exponential in n, and the expected running time of the algorithm of Gramm *et al.* [57] is $O(n\ell + nd \cdot d^{2+\epsilon})$ (Section 4.2).
- We show the applicability of the algorithms for solving Closest String instances in speeding up motif detection (Section 4.3).

4.1 A Linear-Time Algorithm for Solving Small Closest String Instances

In this section, we provide empirical evidence suggesting that the majority of decoy sets have a subset of cardinality four that is a decoy, and describe a linear-time algorithm for determining whether a set of four strings is a decoy, addressing an open problem of Gramm *et al.* [57]. Our empirical results together with our algorithm suggest that the majority of instances of the Closest String problem can be solved in polynomial time.

We begin with some definitions concerning general string analysis. Let ℓ, d, and n be positive integers with $d \leq \ell$. Let σ_i be a function that returns the ith symbol in a string. For any symbol $\beta \in \Sigma$ let β^ℓ denote the length ℓ string of all β symbols. Given a set of strings $S = \{s_1, \ldots, s_n\}$, each of which has length ℓ, the ith *column* refers to the column vector $c_i = [\sigma_i(s_1), \ldots, \sigma_i(s_n)]^T$ in the $n \times \ell$ matrix representation of S.

We will be interested in the cardinality of a decoy set, that is, the number of strings contained in the set. We say set $\hat{S} \subseteq S$ is a decoy of *minimal cardinality* if $\hat{S}$ is a decoy set such that for all $S' \subseteq S$, if $|S'| < |\hat{S}|$, then S' has a center string.

Gramm *et al.* [56] refer to the process of permuting the columns of S such that these are grouped by column type as *normalization*. A normalized instance can be derived from the input set of strings by a simple linear-time algorithm. Given a Closest String for the normalized set of strings, the inverse of this same permutation returns a center string for the original input [56].

Group	Four	Three				Two		
# of columns.	$\lambda_{\beta\beta\beta}$	$\lambda_{\alpha\beta\beta}$	$\lambda_{\beta\alpha\beta}$	$\lambda_{\beta\beta\alpha}$	$\lambda_{\alpha\alpha\alpha}$	$\lambda_{\beta\alpha\alpha}$	$\lambda_{\alpha\beta\alpha}$	$\lambda_{\alpha\alpha\beta}$
s_1	β	α	β	β	α	β	α	α
s_2	β	β	α	β	α	α	β	α
s_3	β	β	β	α	α	α	α	β
s_4	β	β	β	β	β	β	β	β
maj_i	β	β	β	β	α	-	-	-

Table 4.1: Defining the different types of columns for a set of strings with cardinality four. The values $\lambda_{\alpha\beta\beta}$ through $\lambda_{\alpha\alpha\beta}$ denote the number of columns of each type in groups three and two. The symbol "-" implies that the value is undefined at these columns.

4.1.1 Definitions Specific to Sets of Cardinality Four

Given a binary alphabet $\Sigma = \{\alpha, \beta\}$ and a set of strings $S = \{s_1, \ldots, s_4\}$, the symbols in each column have either two, three, or four matching symbols. Sixteen types of columns are possible in general. We say a column belongs to *group* i if it has exactly i matching symbols. To reduce the number of possible types to eight, suppose without loss of generality that $s_4 = \beta^\ell$. Equivalently, create a new set S' by performing a logical exclusive-or of each string in S with s_4 (say α corresponds to boolean true). A center string for S is found by performing another exclusive-or on a center string for S'. Let λ_{abc} denote the number of instances of column $(a, b, c, \beta)^T$, where $a, b, c \in \{\alpha, \beta\}$. See Table 4.1.1. Note that only columns of groups three and two need to be considered, since any center string will correspond to the majority at each column of group four. A pair of columns are considered to be *identical* if a pair of strings in one column mismatch if and only if the same strings mismatch in the second column. For example the column $[\alpha\alpha\beta\beta]^T$ is identical to $[\beta\beta\alpha\alpha]^T$, but neither is identical to $[\alpha\beta\beta\alpha]^T$.

Let maj_i denote the majority of the four symbols in column i. That is, $\text{maj}_i = \alpha$ if symbol α occurs three or more times in column i and $\text{maj}_i = \beta$ if symbol β occurs three or more times; maj_i is undefined if α and β each occur twice. Assuming that $s_4 = \beta^\ell$, only the columns associated with $\lambda_{\alpha\alpha\alpha}$ are such that $\text{maj}_i = \alpha$.

4.1.2 Ubiquitousness or Rarity of Bounded Decoy Sets

We consider the relative frequency, or infrequency, of decoy sets that do not have a proper subset that is also a decoy. Our empirical results demonstrate that the relative frequency of such decoy sets is minimal, and that the majority of decoy sets contain a decoy subset of cardinality four. Still, the results of Gramm *et al.* [57] imply that we cannot characterize

all decoy sets of arbitrary size n as having a proper subset that is a decoy. We refer to a set Q of decoys, each of cardinality n, as having decoys of *bounded cardinality* if every decoy in Q has a proper subset that is a decoy.

Proposition 4.1.1 *Let Σ denote an alphabet of arbitrary fixed size. If $P \neq NP$, then for any n_0 there exists a decoy set S such that every subset of S of cardinality n_0 has a center string.*

Proof Suppose otherwise. That is, there exists an n_0 such that every decoy S of size $n \geq n_0$ has a subset of size n_0 that is a decoy. By Gramm *et al.* [57], for any fixed n_0, there exists an algorithm that decides whether a set of n_0 strings is a decoy in $f(\ell, d)$ time, where $f(\ell, d)$ is polynomial in ℓ and d. Consequently, for any set of $n \geq n_0$ strings S, we can check each of the $\binom{n}{n_0}$ subsets of S of size n_0 to determine whether any is a decoy in time $O(n^{n_0} f(l, d)))$. That is, we can determine in polynomial time whether S is a decoy. Since the Closest String problem is NP-complete, this is possible only if P = NP. □

It should be noted that Proposition 4.1.1 does not preclude that there may exist values of n such that all decoys of cardinality n have a minimal decoy set of cardinality n. Proposition 4.1.1 implies that there does not exist a threshold n_0 such that every decoy set contains a minimal decoy set of cardinality at most n_0, if $P \neq NP$. Although Proposition 4.1.1 implies that no fixed n_0 exists, we conjecture that most decoys have a subset of size four that is a decoy. We provide evidence toward this property with an empirical study on random sets of binary strings which we now describe. In turn, these results motivate the need for an efficient algorithm for determining whether a set of four strings has a center string; we describe such an algorithm in Section 4.1.3.

We empirically investigate the rarity of the occurrence of decoy sets of cardinality n for which the cardinality of a minimal decoy set is large relative to n. We sampled without replacement 1000 times from the set of all possible pairwise-bounded sets of binary, length-ℓ strings; each set sampled has exactly n strings taken from the binary alphabet. We varied the values for n, ℓ and d. For each sample set, we determined whether the set is a decoy or a valid motif, with respect to the value of d, and determined the cardinality of the minimum decoy set. We repeated this experiment ten times and calculated the mean values obtained. Table 4.2 outlines this data. One significant empirical trend demonstrates that as the number of strings increased, the number of decoys that do not contain a minimal decoy of cardinality four became exponentially smaller; when n was ten and twelve the number of minimal decoys of size larger than four was zero. The only value of n for which decoys of size n were seen was six. Further, the total number of decoys in 900,000 sets of strings sampled was approximately 1,500. In summary, the empirical results suggest that a large percentage of binary decoys can be characterized by containing a minimal decoy set of size four, the smallest size possible, further motivating the main results in the following subsection.

Number of strings	$\ell = 8$, $d = 3$			$\ell = 10$, $d = 3$			$\ell = 15$, $d = 4$		
	$n = 6$	$n = 10$	$n = 12$	$n = 6$	$n = 10$	$n = 12$	$n = 6$	$n = 10$	$n = 12$
No. of valid motif	443.4	4.6	88.7	394.4	9.3	3.6	101.4	3.5	2.6
4	542.2	995.4	991.3	605.6	990.7	996.4	898.6	996.5	997.4
5	12.4	0	0	7	0	0	4.1	0	0
6	2	0	0	0.2	0	0	0.8	0	0
7	-	0	0	-	0	0	-	0	0
8	-	0	0	-	0	0	-	0	0
9	-	0	0	-	0	0	-	0	0
10	-	0	0	-	0	0	-	0	0
11	-	-	0	-	-	0	-	-	0
12	-	-	0	-	-	0	-	-	0

Table 4.2: Experimental data illustrating the ubiquitousness of decoy sets of cardinality four. Data obtained from calculating the average of 10 experiments that obtain a random sample, without replacement, of 1000 sets of strings and determining the size of the minimal decoy contained in each set decoy obtained in the sample. The first column is the cardinality of the minimum decoy set.

4.1.3 Finding a Center String for a Set of Four Strings

The only previous polynomial-time solution for finding a center string of a set of four strings was intended more to demonstrate the fixed-parameterized tractability of the problem rather than to provide an efficient solution [57]. As acknowledged by its authors, the corresponding description (for which many details are omitted) results in an algorithm with extremely high (although theoretically linear) run time and, furthermore, does not lend itself well to simple or practical implementation. We present a simple linear-time algorithm for finding a center string of a set of four binary strings or determining that the set is a decoy. After describing the algorithm, we prove its correctness and show its worst-case run time is $O(\ell)$ for any arbitrary d.

Given a set of binary strings $S = \{s_1, \ldots, s_4\}$, algorithm BinaryClosestString4 identifies a center string s^* for S if one exists. Again, to simplify the algorithm's description, suppose $s_4 = \beta^\ell$. The algorithm greedily assigns symbols to s^*, one symbol at a time. Each column c_i is initially considered to be *free*; that is, no symbol has been assigned to $\sigma_i(s^*)$. Once it is assigned a symbol, we say column c_i is *fixed* and its value is not modified again. The algorithm has three phases in which columns of groups four, three, and two are fixed, respectively.

Phase One. Fix symbols of s^* in all columns of group four such that these agree with the symbol of the corresponding column.

Phase Two. The symbols of s^* in columns of group three are fixed sequentially. Say the first $i-1$ columns of group three have been fixed and consider the ith such column. Let s_j denote the string in S that disagrees with the remaining three strings in this column. Let s^+ denote the string given by the symbols of s^* in the fixed columns and the symbols of s_j in the free columns. If s^+ is a center string for S, then let $s^* = s^+$ and return s^*. Otherwise, fix the current column of s^* to agree with the majority and continue to the next column of group three.

Phase Three. If phase three is reached, then only columns of group two remain, of which at most three types may be present. The free columns are fixed by selecting the number of columns of each type that will be assigned symbol α versus β. That is, a solution for columns of group two corresponds to a triple of integers (x, y, z), where $x \in [0, \lambda_{\beta\alpha\alpha}]$ denotes the number of columns of type $\lambda_{\beta\alpha\alpha}$ that will be assigned the symbol α and $\lambda_{\beta\alpha\alpha} - x$ represents the number that will be assigned the symbol β. The variables y and z are defined analogously. See Table 4.3. We denote the corresponding string by $s^*_{x,y,z}$. Therefore, the problem reduces to identifying an integer triple (x, y, z) selected from the region $R = [0, \lambda_{\beta\alpha\alpha}] \times [0, \lambda_{\alpha\beta\alpha}] \times [0, \lambda_{\alpha\alpha\beta}]$ that minimizes

$$f(x, y, z) = \max_{s_i \in S} d(s_i, s^*_{x,y,z}), \tag{4.1}$$

where

$$d(s_1, s^*) = \lambda_{\alpha\beta\beta} + x + \lambda_{\alpha\beta\alpha} - y + \lambda_{\alpha\alpha\beta} - z, \tag{4.2a}$$
$$d(s_2, s^*) = \lambda_{\beta\alpha\beta} + \lambda_{\beta\alpha\alpha} - x + y + \lambda_{\alpha\alpha\beta} - z, \tag{4.2b}$$
$$d(s_3, s^*) = \lambda_{\beta\beta\alpha} + \lambda_{\beta\alpha\alpha} - x + \lambda_{\alpha\beta\alpha} - y + z, \tag{4.2c}$$
$$d(s_4, s^*) = \lambda_{\alpha\alpha\alpha} + x + y + z. \tag{4.2d}$$

The string $s^*_{x,y,z}$ does not actually need to be constructed since the corresponding value of (4.1) is obtained in constant time upon fixing values for x, y, and z.

Instead of evaluating all integer combinations for (x, y, z) (requiring $O(\ell^3)$ time), we identify a set $T \subseteq \mathbb{Q}^3 \cap R$ containing a constant number of triples such that the optimal (possibly non-integer) solution to (4.1) is a triple in T. Interpreted geometrically, (4.2a) through (4.2d) correspond to four respective hyperplanes in $\mathbb{R}^4$ whose maximum, $f(x, y, z)$, defines a surface. Let

$$\begin{aligned}
x_0 &= \frac{1}{4}\left(-\lambda_{\alpha\beta\beta} + \lambda_{\beta\alpha\beta} + \lambda_{\beta\beta\alpha} - \lambda_{\alpha\alpha\alpha} + 2\lambda_{\beta\alpha\alpha}\right), \\
y_0 &= \frac{1}{4}\left(\lambda_{\alpha\beta\beta} - \lambda_{\beta\alpha\beta} + \lambda_{\beta\beta\alpha} - \lambda_{\alpha\alpha\alpha} + 2\lambda_{\alpha\beta\alpha}\right), \\
z_0 &= \frac{1}{4}\left(\lambda_{\alpha\beta\beta} + \lambda_{\beta\alpha\beta} - \lambda_{\beta\beta\alpha} - \lambda_{\alpha\alpha\alpha} + 2\lambda_{\alpha\alpha\beta}\right).
\end{aligned} \tag{4.3}$$

Column Group		Four	Three				Two		
Algorithm Phase		1	2				3		
Number of Columns		$\lambda_{\beta\beta\beta}$	$\lambda_{\alpha\beta\beta}$	$\lambda_{\beta\alpha\beta}$	$\lambda_{\beta\beta\alpha}$	$\lambda_{\alpha\alpha\alpha}$	$\lambda_{\beta\alpha\alpha}$	$\lambda_{\alpha\beta\alpha}$	$\lambda_{\alpha\alpha\beta}$
Set S	s_1	β	α	β	β	α	β	α	α
	s_2	β	β	α	β	α	α	β	α
	s_3	β	β	β	α	α	α	α	β
	s_4	β	β	β	β	β	β	β	β
Center string	s^*	β	β	β	β	α	x	y	z

Table 4.3: The center string found by algorithm BinaryClosestString4 (if one exists) is displayed in the last row. The center string is denoted as s^*. The values $\lambda_{\beta\beta\beta}$ through $\lambda_{\alpha\alpha\beta}$ denote the number of columns of each type and functions x, y, and z denote the number of occurrences of symbol α in the corresponding column as derived by the algorithm.

If $(x_0, y_0, z_0) \in R$, then let $T = \{(x_0, y_0, z_0)\}$. Otherwise, let T denote the set of triples that correspond to x-, y-, and z-coordinates of vertices of the intersection of the surface defined by (4.1) with the boundary of R. If this intersection is empty, then it follows that no center string exists.

For each triple $(x, y, z) \in T$, evaluate the integer triples within unit L_∞ distance of (x, y, z) in region R. That is, for every $(x, y, z) \in T$, consider the integer triples in $[\max(0, x-1), \min(x+1, \lambda_{\beta\alpha\alpha})] \times [\max(0, y-1), \min(y+1, \lambda_{\alpha\beta\alpha})] \times [\max(0, z-1), \min(z+1, \lambda_{\alpha\alpha\beta})]$, of which there are at most eight. Compute (4.1) for each such integer triple (x, y, z) and store the corresponding minimizing string $s^*_{x,y,z}$. Let $s^* = s^*_{x,y,z}$.

Termination. Consider the maximum distance between s^* and a string in S, *i.e.*, the minimum (integer) value of (4.1). If this value is at most d, then s^* is returned as a center string for S. Otherwise, S is a decoy set and no center string exists.

We now demonstrate that algorithm BinaryConsensus4 correctly returns a center string s^* for every set S that is a valid motif set. Furthermore, this is achieved in $O(\ell)$ time, independent of d. The proof of Theorem 4.1.2 refers to Lemmas 4.1.3 and 4.1.4 which follow.

Theorem 4.1.2 *Given any $d \in \mathbb{Z}^+$, any $\ell \in \mathbb{Z}^+$, and any set S of four binary strings of length ℓ, algorithm* BinaryConsensus4 *returns a center string for S with degeneracy parameter d if one exists or returns that S is a decoy in $O(\ell)$ time.*

Proof The correctness of Phase One is straightforward. The correctness of Phase Two follows by induction on i using Lemma 4.1.3. Consequently, if S has a center string, then either one has been found by the end of Phase Two (i.e., $s^* = s^+$), or there exists a center string s^* such that $\sigma_x(s^*) = \text{maj}_x$ for all columns of groups three and four. The optimal

solution for the remaining free columns is found in Phase Three. The correctness of Phase Three follows by Lemma 4.1.4. Therefore, algorithm BinaryClosestString4 returns a center string s^* if one exists, and returns that no center string exists otherwise.

Each phase requires a single pass through the columns of S. Phase One simply requires counting the number of columns of each type. Phase Two also requires maintaining the twelve distances $d(s_i, s_j^+)$ for each $\{i, j\} \subseteq \{1, \ldots, 4\}$, where s_j^+ denotes s^+ for which the free columns are defined according to s_j (as described in Phase Two of algorithm BinaryConsensus4). Every time a column of group three is fixed, each of these twelve values can be updated in constant time. Phase Three simply requires counting the number of columns of each type. Since (4.1) is defined by the maximum of four hyperplanes and region R is bounded by three pairs of parallel planes, the number of triples in T is constant and, furthermore, the coordinates of these triples are straightforward to compute in constant time. Finally, since any point in $\mathbb{R}^3$ has at most eight integer points within unit L_∞ distance from it, the set of integer triples evaluated is also computed in constant time and space. Therefore, algorithm BinaryConsensus4 terminates in $O(\ell)$ time. □

Majority-Rule Property We say property $P(i)$ holds for a set S of four binary strings if and only if either

1. S is a decoy, or
2. there exists a center string for S for which the first i columns of group three have value maj_i.

Lemma 4.1.3 *Let $S = \{s_1, \ldots, s_4\}$ denote a set of four binary strings that has m columns of group three. If $P(i)$ holds for some $i \in \{0, \ldots, m-1\}$ and $s_j \in S$ denotes the string that mismatches in the $(i+1)$st column of group three, then either*

1. *$P(i+1)$ holds for S, or*
2. *s^+ is a center string for S,*

where $\sigma_x(s^+) = \text{maj}_x$ in the first i columns of group three and $\sigma_x(s^+) = \sigma_x(s_j)$ in the remaining columns.

Proof If s^+ is a center string for S then the claim holds. Similarly, if S is a decoy then $P(i+1)$ is true and the claim holds. Therefore, suppose S is not a decoy and s^+ is not a center string for S. By $P(i)$, S has a center string s^* for which the first i columns of group three have value maj_x. Let k denote the index of the $(i+1)$st column of group three.

Case 1. Suppose $\sigma_k(s^*) = \text{maj}_k$. Therefore, $P(i+1)$ is true, and the claim holds.

Case 2. Suppose $\sigma_k(s^*) \neq \text{maj}_k$. That is, $\sigma_k(s^*) = \sigma_k(s_j)$. Since s^+ is not a center string, s^* and s^+ must differ in at least one column; let k' denote the index of such a column. Let s^{**} denote a string of length ℓ such that $\sigma_x(s^{**}) \neq \sigma_x(s^*)$ for $x \in \{k, k'\}$ and $\sigma_x(s^{**}) = \sigma_x(s^*)$ otherwise. That is, $\sigma_k(s^{**}) = \text{maj}_k$. Thus, $d(s_j, s^{**}) = d(s_j, s^*)$ and, furthermore,

$$\forall x \in \{1, \ldots, 4\},\ d(s_x, s^{**}) \leq d(s_x, s^*).$$

Therefore, s^{**} is a center string for S, $P(i+1)$ is true, and the claim holds. □

Lemma 4.1.4 *There exists an integer triple* $(x, y, z) \in [0, \lambda_{\beta\alpha\alpha}] \times [0, \lambda_{\alpha\beta\alpha}] \times [0, \lambda_{\alpha\alpha\beta}]$ *that minimizes* (4.1) *and is within unit* L_∞ *distance from a triple in* T*, where set* T *contains either* (4.3) *or the set of triples that correspond to vertices of the intersection of the surface defined by* (4.1) *with the boundary of* R*.*

Proof Since no two of the hyperplanes induced by (4.2a) through (4.2d) are parallel, $f(x, y, z)$ is a convex function whose surface includes a unique simplicial vertex located at the point of intersection of these four hyperplanes. Furthermore, this point minimizes $f(x, y, z)$ since f is increasing as it tends to infinity in any direction. Thus, $f(x, y, z)$ is minimized at a unique (possibly non-integer) point found by solving for x, y, and z in

$$d(s_1, s^*) = d(s_2, s^*) = d(s_3, s^*) = d(s_4, s^*). \tag{4.4}$$

The constraints of (4.4) corresponds to a system of three linear equations with the unique solution (4.3).

Since the coefficients of x in (4.2a) through (4.2d) are all ± 1, function $f(x, y, z)$ has slope ± 1 along the x-axis for any fixed y and z. The same holds for any fixed x and y or any fixed x and z. Consequently, since $f(x, y, z)$ is convex, a minimum integer solution to (4.1) lies within unit L_∞ distance of its non-integer solution. The set T contains either the unique minimum (4.3) (if it lies within region R) or the set of triples that correspond to vertices of the intersection of the surface defined by $f(x, y, z)$ with the boundary of R, one of which must minimize $f(x, y, z)$ over R. Therefore, the claim holds. □

We have obtained a linear-time algorithm for solving the Closest String decision problem for four binary strings, and considered the relative frequency of decoy sets that do not have a proper subset that is also a decoy. Our results generalize previous work and answer some open problems proposed in [56]. Further, our empirical results demonstrate that the majority of decoy sets contain a decoy subset of cardinality four, suggesting that majority of decoy sets can be characterized as having a decoy set of cardinality four. Such a characterization would allow the Closest String problem to be solved efficiently with high probability.

4.2 Smoothed Analysis of Closest String Instances

Up to this point in this chapter, we have restricted interest to Closest String instances where the cardinality of the instances are small. Now we consider the opposing problem: solving instances that have a large number of strings. Using smoothed analysis, we demonstrate that Closest String instances can be efficiently solved in practice. To the best of our knowledge, this is the first analysis of a natural random model of the Closest String problem. In this section, we give two smoothed analysis results.

Several other papers discuss the smoothed complexity of continuous problems [13, 42] and of discrete problems [7, 76]. The smoothed complexity of other string and sequence problems has been considered by Andoni and Krauthgamer [5], Manthey and Reischuk [80], and Ma [76]. Andoni and Krauthgamer [5] study the smoothed complexity of sequence alignment by the use of a novel model of edit distance; their results demonstrate the efficiency of several tools used for sequence alignment, most notably PatternHunter [78]. Manthey and Reischuk gave several results considering the smoothed analysis of binary search trees [80]. Ma demonstrated that a simple greedy algorithm runs efficiently in practice for Shortest Common Superstring [76].

We use smoothed analysis to prove that the Closest String problem can be efficiently and accurately solved in practice by two different algorirthms, when the number of strings is significantly large. We present two perturbation models, and analyze the expected approximation for *CSP-Greedy* and the expected running time for a well-known, fixed parameter tractability algorithm.

First, we describe our algorithm, *CSP-Greedy*, prove it achieves a 2-approximation guarantee, and give a bound on the probability that *CSP-Greedy* returns an optimal solution to the Closest String instance. We show that *CSP-Greedy* achieves an $1 + f(\epsilon, n, \ell)$-approximation guarantee in time $O(\ell^3)$, where $\epsilon > 0$ is any small value and $f(\epsilon, n, \ell)$ approaches zero in time that is exponential in n.

Next, we explain the good performance of the $O(n\ell + nd \cdot d^d)$-time algorithm by Gramm *et al.* [57] through smoothed analysis. The algorithm is based on an efficient breadth-first search of all possible solutions; their original analysis showed the size of the breadth-first search is bounded by $(d + 1)^d$, however, the empirical analysis suggested that the expected size of the search tree is on the order of $3d$. We give an analytical explanation of these results. Imperative to our analysis is the introduction of a perturbation model of the Closest String model and the probabilistic analysis of the $O(n\ell + nd \cdot d^d)$ time algorithm by Gramm *et al.* [57]. We show for any given Closest String instance I, the average running time of this algorithm on a small perturbation of I is $O(n\ell + nd \cdot d^{2-\epsilon})$.

4.2.1 A Randomized Algorithm for Closest String

Our randomized algorithm, which we refer to as *CSP-greedy*, begins with a majority string s, then successively updates s to make it closer to at least one string in S that does not have s as a center string. We do a maximum number of ℓ cumulative updates to s. After this point, if a center string is not determined, then the process is repeated. If a set of strings S does not have a center string with respect to some value of d, then "not found" will always be returned but if it is a motif set, then with some probability a center string is returned. This probability depends on the number of times we repeat the search. We prove that *CSP-Greedy* achieves a 2-approximation guarantee, and give a bound on the probability that *CSP-Greedy* returns an optimal solution to the Closest String instance.

In [99], Schöning considers the following simple probabilistic algorithm for solving the NP-complete problem of k-SAT: randomly choose a starting assignment and subsequently augment this initial assignment until a satisfying one is obtained. Papadimitriou introduced this random paradigm in the context of 2-SAT and obtained an expected quadratic time bound [85]. These type of algorithms are referred to as Monte Carlo algorithms with one-sided error; a useful property of such algorithm is that the error probability can be made arbitrarily small with repeated independent random repetitions of the search process. We analyze a similar probabilistic approach for the Closest String problem.

CSP-Greedy begins with a string $s_{maj,0}$ randomly selected from all majority strings and iteratively augments the string so that it is closer to one of the strings in S. We let $s_{maj,i}$ be $s_{maj,0}$ after it has been updated i times. At iteration $i+1$, we obtain the string $s_{maj,i+1}$ by augmenting $s_{maj,i}$ so that it has smaller Hamming distance to at least one string s_k in S where $d(s_k, s_{maj,i}) > \Delta d$. This process is repeated ℓ times at which point the process is restarted if there is at least one string s_k in S where $d(s_k, s_{maj,\ell}) > \Delta d$.

Let t be the number of times a random majority string is chosen and augmented ℓ times. Later in the section we define t in terms of the parameters ℓ, n, and d. In order to determine a center string corresponding to the optimal closest distance, this search process is repeated from $\Delta d = 0$ to ℓ.

We give worst-case bounds on the approximation guarantee of *CSP-Greedy*. Also, we present examples of inputs for which the algorithm performs poorly and hence, give lower bounds on the approximation guarantee.

Proposition 4.2.1 *The approximation ratio of the CSP-Greedy algorithm is at most 2 for any alphabet size* $|\Sigma|$.

Proof Since *CSP-Greedy* begins with a randomly selected majority string, it is sufficient to show that for any set of strings S the maximum distance from any majority string to

Algorithm 3 Procedure *augment*

Input: A set S of n ℓ-length strings, parameter d.
Output: A ℓ-length string s or "not found"
Let $\mathcal{S}$ be the set of all ℓ-length majority strings
Select $s_{maj,0}$ randomly from $\mathcal{S}$.
For $i = 0, \ldots, \ell$:
  If $d(s_{maj,i}, s_j) \leq d$ for all $s_j \in S$ then return s and terminate.
  Else $P = \{j : s_j \in S \text{ and } d(s_j, s_{maj,i}) > d\}$
  Choose a p at random from P, and $1 \leq k \leq \ell$ such that $s_p(k) \neq s_{maj,i}(k)$
  Set $s_{maj,i+1}$ to be equal to s_p at position k and equal to $s_{maj,i}$ at all other positions
Return "not found"

Algorithm 4 CSP-Greedy Algorithm

Input: A set S of n ℓ-length binary strings.
Output: A ℓ-length string s
For Δd each from $0 \rightarrow \ell$
  Repeat *augment* t times with parameter Δd

any string in S is twice the optimal closest distance. Let S be a set of strings with optimal closest distance equal to d_{opt}. Without loss of generality assume 0^ℓ is a closest string for S and hence the maximum number of non-zero positions in each string $s_i \in S$ is at most d_{opt}. By the pigeonhole principle, the maximum number of positions containing greater than $n/2$ non-zero positions in a given column is at most $2d_{opt}$. Therefore, any majority string can have at most $2d_{opt}$ non-zero positions and is at distance at most $2d_{opt}$ from each string in S. □

The following example shows the 2-approximation guarantee is tight: $S = \{10000001111, 01000001111, 11111111111, 11111111111\}$, where the majority string is $c_1^* = 11111111111$ and the optimal closest distance is $\ell/4$. On the first iteration the majority string could be altered to $c_2^* = 01111111111$. On the second iteration we could choose to alter this sequence back to c_1^*. It is possible to alternate between c_1^* and c_2^* and end up returning c_1^*, with closest distance $\ell/2$ or twice the optimal.

Probabilistic Analysis of *CSP-Greedy*

The process of augmenting a randomly selected majority string ℓ times or until a closest string is found can be viewed as a Markov chain. This abstraction will be useful in achieving an upper bound on the probability that *CSP-Greedy* returns an optimal solution.

Let a set S be *uniquely satisfiable* if there exists exactly one string s^* where $d(s_i, s^*) \leq d$ for all $s_i \in S$. If S is an instance that does not have a string s such that $d(s, s_i) \leq d^*$ for all $s_i \in S$, then procedure *augment* will return "not found". So we assume otherwise, that the set S is uniquely satisfiable and denote the probability of obtaining s^* when $\Delta d = d^*$ as p_s. If $s_{maj,i}$ is not equal to s^* then there is at least one letter of $s_{maj,i}$ that can be changed so that $d(s_{maj,i}, s^*)$ decreases by one; the probability of this occurring is at least $1/\ell$. Denote $X_i \in \{0, 1, \ldots, \ell\}$ $(i = 0, 1, \ldots)$ as the random variable that is equal to the Hamming distance between $s_{maj,i}$ and s^*, where i is the number of iterations of the *augment* procedure. Each time a position is selected and the value of that position is augmented, either the Hamming distance is increased or decreased by one.

The process $X_0, X_1, X_2, \ldots$ is a Markov chain with a barrier at state ℓ and contains varying time and state dependent transfer probabilities. This process is overly complicated and we instead choose to analyze the following process: $Y_0, Y_1, Y_2, \ldots$, where Y_i is the random variable which is equal to the state number after i steps and there exist infinitely many states. Initially, this Markov chain is started like the stochastic process above (*i.e.* $Y_0 = X_0$). As long as the inner loop is iterating, we let $Y_{i+1} = X_i - 1$ if the process decreases the Hamming distance between $s_{maj,i}$ and s^* by one; and $Y_{i+1} = X_i + 1$ otherwise. After the loop exits, we continue with the same transfer probabilities. By induction on i, it is clear that for each i, $X_i \leq Y_i$, and it follows that p_s is at least $\Pr[\exists t \leq \ell : Y_t = 0]$.

We made the assumption that the set was uniquely satisfiable, however, this assumption is not needed – the random walk may find another center string while not in the terminating state but this possibility only increases the probability the algorithm terminates.

The following asymptotic estimation is used in the proof of the next theorem.

Fact 1 *For $i > 0$ and $j > 0$ the following asymptotic estimation holds:*

$$\sum_{i=0}^{j} \binom{2i+j}{i} \left(\frac{\ell-1}{\ell}\right)^i \left(\frac{1}{\ell}\right)^{i+j} \asymp \left(\frac{1}{\ell-1}\right)^j. \tag{4.5}$$

Proof We set $i = \phi j$ and estimate the summand

$$\binom{2i+j}{i} \left(\frac{\ell-1}{\ell}\right)^i \left(\frac{1}{\ell}\right)^{i+j}$$

by

$$\left(\left(\frac{1+2\phi}{\phi}\right)^{\phi} \left(\frac{1+2\phi}{1+\phi}\right)^{1+\phi} \left(\frac{\ell-1}{\ell}\right)^{\phi} \left(\frac{1}{\ell}\right)^{1+\phi}\right)^j;$$

this holds due to Stirling's formula $n! \asymp (n/e)^n$, giving us:

$$
\begin{aligned}
y &= \left(\frac{1+2\phi}{\phi}\right)^{\phi}\left(\frac{1+2\phi}{1+\phi}\right)^{1+\phi}\left(\frac{\ell-1}{\ell}\right)^{\phi}\left(\frac{1}{\ell}\right)^{1+\phi} \\
\ln y &= \phi\ln\left(\frac{1+2\phi}{\phi}\right)+(1+\phi)\ln\left(\frac{1+2\phi}{1+\phi}\right)+\phi\ln\left(\frac{\ell-1}{\ell}\right)+(1+\phi)\ln\left(\frac{1}{\ell}\right) \\
\frac{1}{y}\cdot\frac{dy}{d\phi} &= \ln\left(\frac{1+2\phi}{\phi}\right)+\frac{2\phi-(1+2\phi)}{1+2\phi}+\ln\left(\frac{1+2\phi}{1+\phi}\right)+\frac{2(1+\phi)-(1+2\phi)}{1+2\phi} \\
&\quad +\ln\left(\frac{\ell-1}{\ell}\right)+\ln\left(\frac{1}{\ell}\right)
\end{aligned}
$$

And hence, we get

$$
\frac{dy}{d\phi}=\ln\left(\frac{1+2\phi}{\phi}\cdot\frac{1+2\phi}{1+\phi}\cdot\frac{\ell-1}{\ell}\cdot\frac{1}{\ell}\right)\left(\frac{1+2\phi}{\phi}\right)^{\phi}\left(\frac{1+2\phi}{1+\phi}\right)^{1+\phi}\left(\frac{\ell-1}{\ell}\right)^{\phi}\left(\frac{1}{\ell}\right)^{1+\phi}.
$$

Finding the extreme points by setting $\frac{dy}{d\phi}$ equal to zero gives:

$$
0=\ln\left(\frac{1+2\phi}{\phi}\cdot\frac{1+2\phi}{1+\phi}\cdot\frac{\ell-1}{\ell}\cdot\frac{1}{\ell}\right)\left(\frac{1+2\phi}{\phi}\right)^{\phi}\left(\frac{1+2\phi}{1+\phi}\right)^{1+\phi}\left(\frac{\ell-1}{\ell}\right)^{\phi}\left(\frac{1}{\ell}\right)^{1+\phi}.
$$

Since $\ln(x)=0$ when $x=1$, we get:

$$
0=(4(\ell-1)-\ell^2)\phi^2+(4(\ell-1)-\ell^2)\phi+(\ell-1).
$$

Using the quadratic formula we obtain $\phi=1/(\ell-2)$. Hence, the left side of Equation 4.5 is at least equal to $\left(\frac{1}{\ell-2}\right)^j$. □

Theorem 4.2.2 *Let S be a set of strings and d be the optimal closest distance. Then the probability of procedure "augment" obtaining a center string for S is at least $e\cdot 2^{-\ell}$ for sufficiently large ℓ.*

Proof Suppose s^* is a center string for S and let k be the Hamming distance between $s_{maj,0}$ and s^*. Denote $\Pr[Y_{i+1}=j-1|Y_i=j]$ by q_i. Given that the Markov chain starts in some state j, it can reach a halting state in at least j steps by making transitions through the states $j-1$, $j-2$, …, $1, 0$. The probability of this happening is at least $\sum_{i=0}^{j} q_i$. For $i=0,1,2,\ldots$ the halting state can be reached after $2i+j$ steps, where i steps are "bad" and $j+i$ steps are "good". The probability of this happening is:

$$\Pr[Y_{2i+j} = 0 \text{ and } Y_k > 0,\ \forall\, k < 2i + j \,|\, Y_0 = j],$$

which is at least $q_i^{i+k}(1-q_i)^i$ times the number of ways of arranging i bad steps and $i+j$ good steps such that the sequence starts in state j, ends in state 0, and does not reach 0 before the last step. Using the Ballot theorem we know there are $\binom{2i+j}{i}\frac{i}{2i+j}$ possible arrangements of these i and $i+j$ steps. Therefore, the above probability is at least:

$$\binom{2i+j}{i}\frac{i}{2i+j}(1-q_i)^i q_i^{i+j}.$$

This expression is not defined in the case $i = j = 0$. In this case, the probability is equal to 1. Thus, we have:

$$\begin{aligned}
\Pr[Y_{2i+j} &= 0 \text{ and } Y_k > 0\ \forall\, k < 2i + j \,|\, Y_0 = j] \\
&\geq \sum_{j=0}^{\ell} 2^{-\ell}\binom{\ell}{j} \sum_{2i+j\leq c\ell} \binom{2i+j}{i}\frac{i}{2i+j}\left(\frac{\ell-1}{\ell}\right)^i\left(\frac{1}{\ell}\right)^{i+j} \\
&\geq 2^{-\ell}\sum_{j=0}^{\ell}\binom{\ell}{j}\sum_{i=0}^{j}\binom{2i+j}{i}\left(\frac{\ell-1}{\ell}\right)^i\left(\frac{1}{\ell}\right)^{i+j}
\end{aligned}$$

Using Fact 1, we have:

$$\begin{aligned}
\Pr[Y_{2i+j} = 0 \text{ and } Y_k > 0\ \forall\, k < 2i + j \,|\, Y_0 &= j] \\
&\asymp 2^{-\ell}\sum_{j=0}^{\ell}\binom{\ell}{j}\left(\frac{1}{\ell-1}\right)^j \\
&= \left(\frac{\ell}{2(\ell-1)}\right)^{\ell} \text{ [by the Binomial theorem]} \\
&\asymp e\cdot 2^{-\ell}\left(\frac{\ell}{\ell-1}\right) \text{ [by Taylor's theorem]}
\end{aligned}$$

For sufficiently large ℓ we have the probability is at least $e \cdot 2^{-\ell}$. □

The following corollary, an immediate consequence of the previous lemma, bounds the probability of error.

Corollary 4.2.3 *If $t = \ell$ and ℓ is sufficiently large, CSP-Greedy has time complexity $O(\ell^3)$ and one-sided error probability of no more than $\left(1 - e \cdot 2^{-\ell}\right)^{\ell}$.*

Smoothed Analysis of *CSP-Greedy*

We use smoothed analysis to show that the expected approximation ratio of *CSP-Greedy* is $\left(1+\frac{\epsilon(e-1)}{2^n}\right)^{\ell}$. This result explains why this algorithm performs well in practice on instances with significantly large cardinality.

A *perturbed instance* S is defined to be $S' = \{s'_1, s'_2, \ldots, s'_n\}$, where each s'_i has length ℓ and each s'_i is obtained by mutating uniformly at random each letter of s_i with a small probability $p > 0$. In more general terms, an adversary chooses n length-ℓ strings from the binary alphabet, and every symbol is perturbed independently with a small probability p. Next, let S be a Closest String instance, S' be the corresponding perturbed instance, and q be $\Pr[s_i(j) = 1]$. We have

$$\begin{aligned} Pr[s'_i(j) = 0] &= \Pr[s_i(j) = 1]\Pr[s_i(j) \text{ was permuted}] + \\ &\quad \Pr[s_i(j) = 0]\Pr[s_i(j) \text{ was not permuted}] \\ &= qp + (1-q)(1-p). \end{aligned}$$

Assuming $\epsilon > 0$ is a small number, and the perturbation probability satisfies $\frac{\epsilon \log(\ell n)}{\ell n} \leq p \leq \frac{1}{2}$, we obtain the following:

$$\begin{aligned} \Pr[s'_i(j) = 0] &= 1 - q - p + 2qp && (4.6) \\ &\geq 1 - q - p + q \quad \text{since } p \leq 1/2 && (4.7) \\ &\geq \frac{\epsilon \log(\ell n)}{\ell n} && (4.8) \end{aligned}$$

The next theorem, our main result, demonstrates that for significantly large ℓ, as n (and to a lesser extent ℓ) increases the approximation ratio approaches 1. This result explains our experimental results analytically.

Theorem 4.2.4 (*CSP-Greedy* under limited randomness) *For any given small $\epsilon > 0$, for perturbation probability $\frac{\epsilon \log(\ell n)}{\ell n} \leq p \leq \frac{1}{2}$ and significantly large ℓ, the expected ratio of 'CSP-Greedy' on the perturbed instances is $\left(1+\frac{\epsilon e}{2^n}\right)^{\ell}$.*

Proof Given a Closest String instance S, we define the instance as *good* if each majority string of S is also a center string for S; otherwise, we define the instance as *bad*. Since each iteration of procedure *augment* begins by selecting a random majority string and

determining whether it has distance at most d from each string in S, it follows that *CSP-Greedy* will return a center string when S is a good instance. Let p_{bad} be the probability that the perturbed instance is bad.

In order to bound p_{bad}, we calculate the probability that a majority string does not match the center string at a particular position. Without loss of generality assume that 0^ℓ is a center string; we know there exists at least one such string. We calculate the probability that 0^ℓ is a majority string. Let $X_{i,j}$ be a binary random variable representing the symbol of s_i at the j-th position. For a given j, let the number of ones be $X_j = \sum_i X_{i,j}$.

$$\begin{aligned}\Pr[X_j > n/2] &= \sum_{i=\lfloor n/2 \rfloor + 1}^{n} \binom{n}{i} (\Pr[s'_i(j) = 1])^i (1 - \Pr[s'_i(j) = 1])^{n-i} \\ &\geq 1 - (1 - \Pr[s'_i(j) = 1])^n\end{aligned}$$

Let $\alpha = 1 - (1 - \Pr[s'_i(j) = 1])^n$. We give a lower bound for the probability that the instance is good.

$$\begin{aligned}1 - p_{bad} &= 1 - \Pr[S' \text{contains at least one bad column}] \\ &= 1 - \sum_{i=1}^{\ell} \binom{\ell}{i} \Pr[X_j > n/2]^i \\ &\geq 1 - \sum_{i=1}^{\ell} \binom{\ell}{i} \alpha^i \\ &\geq 1 - \alpha^\ell \text{ [by Binomial Theorem]}\end{aligned}$$

Therefore, it follows that p_{bad} is at most $(1 - (1 - \Pr[s'_i(j) = 1])^n)^\ell$ and hence, we get:

$$p_{bad} \leq (1 - (1 - \Pr[s'_i(j) = 1])^n)^\ell \leq \left(1 - \frac{\epsilon}{2^n}\right)^\ell$$

In Theorem 4.2.1 the greedy algorithm was proved to have a worst-case approximation ratio of 2. Therefore, by Theorem 4.2.1 and Corollary 4.2.3 we obtain the following:

$$\begin{aligned}E[ratio] &\leq 2 \left(1 - \frac{\epsilon}{2^n}\right)^\ell \left(1 - \frac{e}{2^\ell}\right)^\ell \text{ [by Corollary 4.2.3]} \\ &\leq 2 \left(1 + \frac{\epsilon e}{2^{n+\ell-1}} - \frac{e}{2^\ell} - \frac{\epsilon}{2^n}\right)^\ell\end{aligned}$$

For significantly large values of n and ℓ, we have:

$$\left(1 + \frac{\epsilon e}{2^{n+\ell-1}} - \frac{e}{2^{\ell}} - \frac{\epsilon}{2^{n}}\right)^{\ell} \leq \frac{1}{2^{1/\ell}} \left(1 + \frac{\epsilon e}{2^{n}}\right)^{\ell},$$

and therefore,

$$E[ratio] \leq \left(1 + \frac{\epsilon e}{2^{n}}\right)^{\ell}.$$

□

We note that our perturbation is very small compared to the size of the instance. With perturbation probability $p = \frac{\epsilon \log(\ell n)}{\ell n}$, each set of n strings of length ℓ is expected to change by $\log(\ell n)$ letters.

4.2.2 Bounded Search Tree Algorithm

In addition to worst-case theoretical analysis, Gramm *et al.* [57] gave experimental results that suggest the $O(n\ell + nd \cdot d^{d})$-time algorithm is more efficient in practice than the worst-case running time. The worst-case analysis relies on showing that the size of the binary search tree is at most $(d+1)^{d}$, whereas, the experiment results demonstrate that the search trees are by far smaller than this bound predicts (*i.e.* in the range of size d). Although, the empirical results are compelling and suggest the practicality of this algorithm, there currently does not exist an analytical reason for the efficiency of the algorithm.

We consider a slight modification of this algorithm by Gramm *et al.* [57]. Rather than beginning with a random string (as in the original algorithm) we will assume that the algorithm begins with a majority string. Our main contribution demonstrates that for sufficiently large values of n and ℓ, the modified algorithm builds a search tree with expected size $d^{2+\epsilon}$, where ϵ is a small value greater than zero. We introduce a new perturbation model for the Closest String instances, and prove that the average running time of the algorithm of Gramm *et al.* [57] is $O(n\ell + nd \cdot d^{2+\epsilon})$ for any given perturbed instance. This analysis involves investigating typical properties of a smoothed instance, including providing a bound for the probability that any string that contains the majority alphabet symbol at each position is a center string. From a practical perspective, these results explain why this fixed parameter algorithm for the Closest String problem performs well in practice.

Gramm *et al.* [57] applied a well-known bounded search tree paradigm, and showed the Closest String problem can be solved in linear time for constant d. The original algorithm of Gramm *et al.* [57] begins by initializing a parameter Δd to d and candidate string s to s_1. Every recursive call decreases Δd by one. The algorithm halts when $\Delta d < 0$.

Therefore, the algorithm builds a search tree of height at most d. At each iteration level of the recursion, a string s_i is chosen from the set of strings that do not have s as a center string, and all $d+1$ of the positions where s and s_i disagree are considered (there are at most $2d$ positions). For each of these positions s is set equal to s_i. This yields an upper bound of $(d+1)^d$ on the search tree size.

Algorithm 5 CSP Bounded Search Algorithm

Input: A Closest String instance I, a candidate string s, and a parameter Δd.
Output: A center string s if it exists, and "not found" otherwise.
If $\Delta d < 0$, then return "Not found"
Choose $i \in \{1, \ldots, n\}$ such that $d(s, s_i) > d$.
    $\mathcal{P} = \{p | s[p] \neq s_i[p]\}$;
    Choose any $\mathcal{P}' \subseteq \mathcal{P}$ with $|\mathcal{P}'| = d+1$.
    For each position $p \in \mathcal{P}'$
        Let $s(p) = s_i(p)$
        s_{ret} = **CSP Bounded Search Algorithm** (s, $\Delta d - 1$)
        If $s_{ret} \neq$ "not found ", then return s_{ret}
Return "not found"

The algorithm we consider initializes the candidate string to be a random, majority vote string (rather than s_1). For any "yes" instance of the Closest String problem the majority vote string s has distance at most $2d$ from each string in S. It follows that the proof of the running time and correctness of the original algorithm continues to hold for our modified algorithm and therefore, the following theorem holds for our modification.

Theorem 4.2.5 *[57] "CSP Bounded Search Algorithm" solves the* Closest String *problem in time* $O(n\ell + nd \cdot d^d)$.

Smoothed Height of the Search Tree

We show *CSP Bounded Search Algorithm* has expected running time $O(n\ell + nd \cdot d^3)$, a result that gives substantial improvement to the worst-case analysis of Gramm *et al.* [57]. Our proof relies on considering the smoothed height of the binary search tree that is obtained by randomly perturbing a given (adversarial) instance, then taking the expected height of the search tree generated by this instance.

We define a new perturbation model for the Closest String problem. We need a perturbation model that will conserve the 'yes' instances; if $I = (S, d)$ is a Closest String instance that contains at least one center string, then after perturbing S and d the resulting instance must also have at least one center string. Let $I = (S, d)$ be a Closest

STRING instance that has at least one center string and without loss of generality, assume $0 \in \Sigma$ and 0^ℓ is a center string for S. A *perturbed instance* of I is defined to be $I' = (S', d')$, where $d' = d + p(\ell - 2d)$ and each string s'_i is obtained by mutating s_i with a small probability $p > 0$ as follows:

- For each j such that $s_i(j) \neq 0$, we let $s'_i(j) = 0$ with probability $p > 0$. Let $\rho_i = \{j | s_i(j) \neq 0 \text{ and } s'_i(j) = 0\}$.
- Select $\eta_i \in (0, d' - (d(s_i, 0^\ell) - \rho_i))$ and choose η_i positions uniformly at random from the set of positions where s_i is equal to 0. For each letter $j \in \eta_i$, we select uniformly at random a symbol in $\Sigma/\{0\}$ to be equal to $s'_i(j)$.

We note that if 0^ℓ is a center string for S with respect to the parameter d then it is a center string for S' with respect to the parameter d'. For the remainder of this section – unless stated otherwise – without loss of generality, if $I = (S, d)$ contains a center string, then $0 \in \Sigma$ and 0^ℓ is a center string.

We define a perturbed instance $I' = (S', d')$ as *simple* if all strings in S' have Hamming distance at most d' from each of the majority strings for S'. This definition is used in our final smoothed analysis and is crucial for obtaining a tight bound on the number of instances in which the algorithm performs poorly. Instances that are simple have the property that the algorithm immediately halts with a correct solution and, therefore, we are assured that the running time of the algorithm is efficient. We refer to a column as *good* if it contains more zeros than nonzeros and, thus, guarantees that the majority symbol is equal to the center string at that position; all other columns are *bad*.

Lemma 4.2.6 *Let $I' = (S', d')$ be a perturbed instance with probability $0 < p \leq \frac{1}{2}$. Then the probability that I' is not simple is at most $1 - \left(1 - \frac{1}{2^n}\right)^\ell$.*

Proof We first calculate the probability that a column is good. Let $X_{i,j}$ be a binary random variable that is equal to 1 if s_i is equal to the value of the center string (*i.e.* equal to 0) at the j-th position. For a given j, let the number of ones be $X_j = \sum_i X_{i,j}$.

$$\begin{aligned}
\Pr[X_j > n/2] &= \sum_{i=n/2}^{n} \binom{n}{i} \left(\Pr[s'_i(j) = 0]\right)^i \left(1 - \Pr[s'_i(j) = 0]\right)^{n-i} \\
&= (1 - \Pr[s'_i(j) = 1])^n \sum_{i=0}^{n} \binom{n}{i} \left(\frac{\Pr[s'_i(j) = 1]}{1 - \Pr[s'_i(j) = 1]}\right)^i \\
&\quad - (1 - \Pr[s'_i(j) = 1])^n \sum_{i=0}^{n/2-1} \binom{n}{i} \left(\frac{\Pr[s'_i(j) = 1]}{1 - \Pr[s'_i(j) = 1]}\right)^i
\end{aligned}$$

It follows from this last equation that:

$$1 - \Pr[s'_i(j) \neq 0]^n \leq \Pr[X_j > n/2] \leq 1 - \Pr[s'_i(j) \neq 0]^{n/2+1}. \tag{4.9}$$

To determine the probability that I' is not simple we calculate the probability 0^ℓ is a majority string and hence, that I' contains at least one bad column.

$$\begin{aligned}
\Pr[I' \text{ is not simple}] &= 1 - \Pr[X_j > n/2 \mid j \in [0, \ell]] \\
&\leq 1 - (1 - \Pr[s'_i(j) \neq 0]^n)^\ell \quad \text{by Equation 4.9} \\
&\leq 1 - \left(1 - \left(\frac{d'}{\ell}\right)^n\right)^\ell \\
&\leq 1 - \left(1 - \frac{1}{2^n}\right)^\ell
\end{aligned}$$

□

We now prove an upper bound for the expected height of the bounded search tree produced by *CSP Bounded Search Algorithm* under the perturbation model we defined previously in this subsection. Before giving this result we prove an important property for estimating the smoothed height of the search. More specifically, we give an upper bound for the probability that the size of the search tree exceeds a specific value.

There are $O(d^d)$ possible paths in the search tree $\mathcal{T}$ corresponding to *CSP Bounded Search Algorithm*. Let P_i be the random indicator variable describing whether the ith path in $\mathcal{T}$ is does or does not result in a center string, *i.e.* let $P_i = 1$ if the ith path results in a center string and $P_i = 0$ otherwise. The algorithm halts when $P_i = 1$. Let $\mathcal{P}$ be the number of paths considered until $P_i = 1$.

Lemma 4.2.7 *Let $\epsilon > 0$ be a small number and $0 < p \leq \frac{1}{2}$. Let $I' = (S', d')$ be a* Closest String *instance that is not simple, then when ℓ is significantly large*

$$\Pr[\mathcal{P} \geq d^{dpc}] \leq \frac{1}{d^{dpc}}, \text{ for any constant } c > 0.$$

Proof If $I' = (S', d')$ is a "no" Closest String instance then the result of each recursive iteration of *CSP Bounded Search Algorithm* will always return false. If S' contains a center string, with some probability, we have $P_i = 1$. Clearly, the expected number of paths in

$\mathcal{T}$ that need to be considered before encountering a path that leads to a center string is $1/\Pr[P_i = 1]$. We now calculate $\Pr[P_i = 1]$. Suppose $I' = (S', d')$ contains a center string and from our previous assumption, we have that $0 \in \Sigma$ and 0^ℓ is a center string.

If the candidate string s in *CSP Bounded Search Algorithm* is not equal to 0^ℓ then there is at least one letter of s that can be changed so that $d(s, 0^\ell)$ decreases by one; the probability of this occurring is at least $1/\ell$. Denote $Y_k \in \{0, 1, \ldots, d'\}$ $(k = 0, 1, \ldots)$ as the random variable that is equal to the Hamming distance between s and 0^ℓ, where k is the number of recursive iterations. Each time a position is selected and the value of that position is augmented, either the Hamming distance is increased or decreased by one.

The process $Y_0, Y_1, Y_2, \ldots$ is a Markov chain with a barrier at state d' and contains varying time and state dependent transfer probabilities. This process is overly complicated and we instead choose to analyze the following process: $Z_0, Z_1, Z_2, \ldots$, where Z_k is the random variable which is equal to the state number after k recursive steps and there exist infinitely many states. Initially, this Markov chain is started like the stochastic process above (*i.e.* $Z_0 = Y_0$). As long as the inner loop is iterating, we let $Z_{k+1} = Y_k - 1$ if the process decreases the Hamming distance between s and 0^ℓ by one, and $Z_{k+1} = Y_k + 1$ otherwise. After the algorithm halts, we continue with the same transfer probabilities. By induction on k, it is clear that for each k, $Y_k \leq Z_k$, and it follows that $\Pr[P_i = 1]$ is at least $\Pr[\exists t \leq d : Z_t = 0]$.

We made the assumption that S' contains only one center string; however, this assumption is not needed – the random walk may find another center string while not in the terminating state but this possibility only increases the probability that the the algorithm terminates.

Given that the Markov chain starts in some state k, it can reach a halting state in at least k steps by making transitions through the states $k-1$, $k-2$, …, 1, 0. The probability of this happening is $(1/\ell)^k$. Also, for $j = 1, 2, 3, \ldots$ the state can be reached in $2j + k$ steps where there are j steps which increase the state number and $k + j$ steps which decrease the state number. Let $q(j, k)$ be the probability that $Z_{2j+k} = 0$, such that the state 0 is not reached in any earlier step, under the condition that the Markov chain started in state k. More formally,

$$q(j, k) \;=\; \Pr[Y_{2j+k} = 0, \;\text{ and }\; Z_\alpha > 0 \,\forall\, \alpha < 2j + k \mid Z_0 = k].$$

Clearly, $q(0, k) = (1/\ell)^k$, In the general case, $q(j, k)$ is $((\ell-1)/\ell)^j (1/\ell)^{j+k}$ times the number of ways of arranging j bad steps and $j+k$ good steps such that the sequence starts in state k, ends in state 0 and does not reach 0 before the last step.

Using the ballot theorem [1] we know there are $\binom{2j+k}{j} \frac{j}{2j+k}$ possible arrangements of

these j and $j+k$ steps. Therefore, the above probability is at least

$$\binom{2j+k}{j}\frac{j}{2j+k}\left(\frac{\ell-1}{\ell}\right)^{j}\frac{1}{\ell^{j+k}}.$$

This expression is not defined in the case $j=k=0$. In this case, the probability is equal to 1.

The following fact, which is easily proved, is used in the next theorem.

Fact 2 *Let $I=(S,d)$ be a* Closest String *instance and s_{maj} be any majority string for S then $d(s^*,s_{maj})\leq 2d$ for any a center string s^* for S.*

Proof We assume n is even. Suppose otherwise that there exists an instance $I=(S,d)$ where one of the majority strings s_{maj} has distance $2d+1$ from the center string s^*. Without loss of generality we assume $0,1\in\Sigma$, 0^ℓ is the center string, and $1^{2d+1}0^{\ell-2d-1}$ is s_{maj}. Thus, there exist at least $n/2$ strings equal to 1 at the first $2d$ positions, implying each string in S has d positions equal to 1. Since each string has distance at most d from 0^ℓ this contradicts the fact that $d(s^*,s_{maj})>2d$. An identical proof can be given for the case when n is odd. □

To calculate $\Pr[P_i=1]$ we need to calculate the probability that k bad columns exist, for $k=1,\ldots,2d$. We let X_{bad} be the total number of columns that are bad in S'. It follows, from Equation 4.9 and the linearity of expectation, that the expected number of bad columns is $\ell(1-\Pr[s'_i=0]^n)$. Therefore we get the following:

$$\begin{aligned}
\Pr[P_i &= 1] \geq \sum_{j=1}^{2d}\Pr[X_{bad}=j]\sum_{i=0}^{j}q(i,j)\\
&\geq (\Pr[X_{bad}<2d+1]-Pr[X_{bad}=0])\sum_{j=1}^{2d}\sum_{i=0}^{j}q(i,j)\\
&= (1-\Pr[X_{bad}\geq \ell-2d-1]-Pr[I' \text{ is simple}])\sum_{j=1}^{2d}\sum_{i=0}^{j}q(i,j)
\end{aligned}$$

It follows from Markov's inequality that:

$$\Pr[P_i=1]\geq\left(1-\frac{\ell(1-\Pr[s'_i(j)=0]^n)}{\ell-2d-1}-Pr[I' \text{ is simple}]\right)\sum_{j=1}^{2d}\sum_{i=0}^{j}q(i,j).$$

By Lemma 4.2.6, we get:

$$\begin{aligned}
\Pr[P_i = 1] &\geq \left(1 - \frac{\ell(1 - \Pr[s_i'(j) = 0]^n)}{\ell - 2d - 1} - \left(1 - \frac{1}{2^n}\right)^\ell\right) \sum_{j=1}^{2d} \sum_{i=0}^{j} q(i,j) \\
&\geq \left(1 - \frac{\ell}{\ell - 2d - 1} - \left(1 - \frac{1}{2^n}\right)^\ell\right) \sum_{i=0}^{2d} q(i,j) \\
&\geq \left(1 - \frac{\ell}{\ell - 2d - 1} - \left(1 - \frac{1}{2^n}\right)^\ell\right) \sum_{i=1}^{2d-1} \left(\frac{1}{1-\ell}\right)^{i+1} \quad \text{[Fact 1]} \\
&\geq \left(1 - \frac{\ell}{\ell - 2d - 1} - \left(1 - \frac{1}{2^n}\right)^\ell\right) \frac{1 - \left(\frac{1}{\ell-1}\right)^{2d+1}}{1 - \frac{1}{\ell-1}}
\end{aligned}$$

Hence, for sufficiently large ℓ we have $\Pr[P_i = 1] = 1 - \left(1 - \frac{1}{2^n}\right)^\ell$ and it follows that:

$$E[\mathcal{P}] = \frac{1}{1 - \left(1 - \frac{1}{2^n}\right)^\ell}$$

and by Markov's inequality we have that for any $c > 0$

$$\Pr[\mathcal{P} \geq d^{dcp}] \leq \frac{1}{d^{dcp}\left(1 - \left(1 - \frac{1}{2^n}\right)^\ell\right)}.$$

Hence, $\Pr[\mathcal{P} \geq d^{dcp}]$ is equal to $\frac{1}{d^{dcp}}$ for significantly large n. □

The following is our main theorem which provides an upper bound on $E[\mathcal{P}]$, the expected size of the search tree. An important aspect of this result is the small perturbation probability required in comparison to the instance size. For a string of length ℓ, the expected number of positions to change is approximately $O(\log n)$.

Theorem 4.2.8 *For any small $0 < \epsilon \leq d - 2 - 3\log d$ and perturbation probability*

$$\frac{\epsilon}{\alpha}\left(\frac{\log d}{d}\right)^3 \leq p \leq \frac{\epsilon}{\alpha}\left(\frac{\log d}{d}\right)^2,$$

where $\alpha \geq \frac{d-2-\epsilon}{\epsilon \cdot 3\log d}$, the expected running time of "CSP Bounded Search Algorithm" on the perturbed instances is $O(n\ell + nd \cdot d^{2+\epsilon})$ for sufficiently large ℓ and n.

Proof There are $O(d^d)$ possible paths in the search tree corresponding to *CSP Bounded Search Algorithm* [57]. The size of the bounded search tree is equal to zero for simple instances and therefore, we are only required to consider the non-simple instances in which Lemma 4.2.7 holds and ones in which it does not. For the former instances we will use Lemma 4.2.7 with $c = \alpha \cdot d$ to bound the expected size of the search tree and for the latter instances we use Lemma 4.2.6. Lemma 4.2.6 states that the probability that an instance is not simple is at most $1 - \left(1 - \frac{1}{2^n}\right)^\ell$. Hence, we have the following bound

$$\begin{aligned}
E[\mathcal{P}] &\leq d^{\alpha \cdot d^2 p} \cdot \Pr[I' \text{ is not simple}] + d^d \cdot \Pr[\mathcal{P} \geq d^{dcp}] \\
&\leq d^{\alpha \cdot d^2 p} \cdot \left(1 - \left(1 - \frac{1}{2^n}\right)^\ell\right) + d^d \cdot \Pr[\mathcal{P} \geq d^{dcp}] \text{ by Lemma 4.2.6} \\
&\leq d^{\alpha \cdot d^2 p} \cdot \left(1 - \left(1 - \frac{1}{2^n}\right)^\ell\right) + d^{d - dcp} \text{ by Lemma 4.2.7} \\
&\leq \frac{d^{\alpha \cdot 2 \log d}}{2^{n\ell}} + d^{d - dcp}
\end{aligned}$$

We consider the first term of the last equation. We have $d^{\alpha \cdot 2 \log d} \leq 2^{n\ell}$ if and only if $\frac{4}{3 \log 2} \leq 5 \leq \frac{\epsilon \cdot n\ell}{d - 2 - \epsilon}$. Therefore, for sufficiently large ℓ and n we have:

$$E[\mathcal{P}] \leq o(1) + d^{d(1 - \frac{\alpha p}{d})}.$$

Next, we show the exponent of the second term is at most $2 + \epsilon$ as follows:

$$\begin{aligned}
1 - \frac{\alpha p}{d} &\leq \frac{2 + \epsilon}{d} \\
d - \epsilon \cdot 3 \log d \cdot \alpha &\leq 2 + \epsilon \\
d - \epsilon \cdot 3 \log d \cdot \frac{d - 2 - \epsilon}{\epsilon \cdot 3 \log d} &\leq 2 + \epsilon
\end{aligned}$$

Therefore, we have $E[\text{search tree size}] \leq o(1) + d^{2+\epsilon}$. The analysis of Gramm *et al.* [57] demonstrated that each recursive step takes time $O(nd)$ and the preprocessing time takes $O(n\ell)$ and therefore, we obtain an overall running time of $O(n\ell + nd \cdot d^{2+\epsilon})$. □

We note that we require ϵ to be at most $d - 2 - 3 \log d$ in order to have $p \in [0, 1]$ and $\frac{\epsilon}{\alpha} \cdot \left(\frac{\log d}{d}\right)^3 \leq \frac{\epsilon}{\alpha} \left(\frac{\log d}{d}\right)^2$. Also, as shown in the previous chapter and by Gramm *et al.* [56],

there exists an efficient solution for the Closest String problem when at least one of n, ℓ or d is reasonably small and therefore, the constraint that n and ℓ are sufficiently large is a reasonable assumption.

4.3 Experimental Results

We performed tests on a PC with a 64-bit 2600 MHz processor and 1 GB of RAM running Ubuntu. For all experiments in this section, we generated synthetic motif sets as follows: a string s is selected at random from the set of all 4^ℓ possible strings, then n motif instances are chosen at random from the set of all strings of distance at most d from s.

4.3.1 Empirical Difference Between a Random String and a Majority String

First, we give empirical evidence for the concept that beginning with a majority string rather than a random string improves upon the efficiency of *CSP-Greedy*. We fixed ℓ to be 15, d to be 4, and varied the value of n from 12 to 36. We generated 100 motif sets for each set of values of ℓ, d and n. For each motif set procedure *augment* of *CSP-Greedy* was run twice – once with $s_{maj,0}$ initialized to be equal to a majority string, and a second time with $s_{maj,0}$ initialized to be equal to a random string – and in both cases, we counted the number of augmentations required to obtain a center string. Figure 4.1 illustrates the mean number of augmentations for 100 motif sets.

Figure 4.1 demonstrates that the number of augmentations required to obtain a center string is significantly larger if $s_{maj,0}$ is initialized to be a random string, rather than a majority string. Further, as the value of n increases the disparity between the number of required number of augmentations of a majority string to obtain a center string and the required number of augmentations of a random string to obtain a center string increases substantially. In particular, when n is equal to 24 the number of augmentations required of the majority string is equal to 0 (as seen in Figure 4.1) – illustrating that the majority string is a center string for all motif sets considered. We observe that this fact cannot be extended to be true for random strings.

Next, we considered the change in the Hamming distance as the values n, ℓ, and d varied. We considered three different value of ℓ and d, varied the value of n from 5 to 25, generated 100 random motif instances, and determined the mean Hamming distance between a majority string and a center string. Figure 4.2 illustrates this data and demonstrates a drastic decrease in the Hamming distance between the majority string and the center string as n increases, for all the (ℓ, d)-motif problems considered. When the value

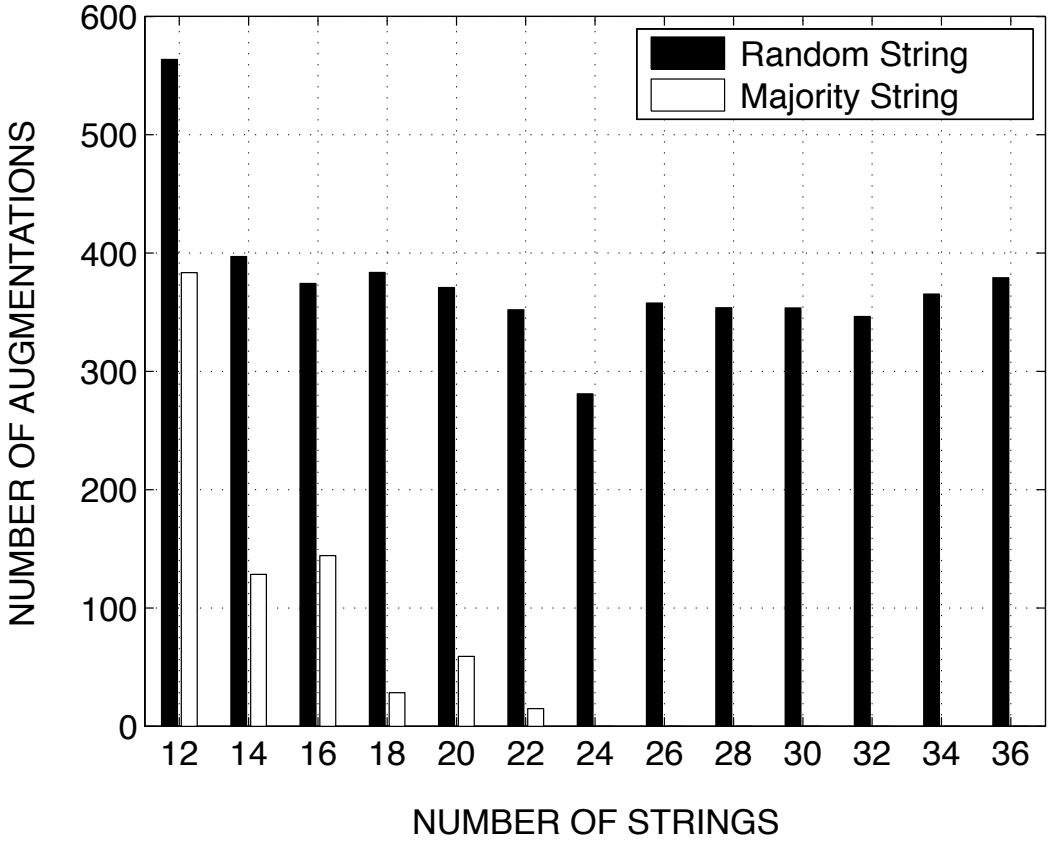

Figure 4.1: A comparison between the number of augmentations required to obtain a center string when procedure *augment* begins with a random string (shown in black), and that required when procedure *augment* begins with a majority string (shown in white), for various values of n. In all experiments, we have ℓ and d equal to 15 and 4, respectively.

of n is significantly large, the distance between the majority string and the center string is equal to zero – implying the majority string is a center string.

Hence, the experimental results in this subsection illustrate the following trend: regardless of the value of ℓ and d considered, for significantly large values of n the majority string is likely to be a center string.

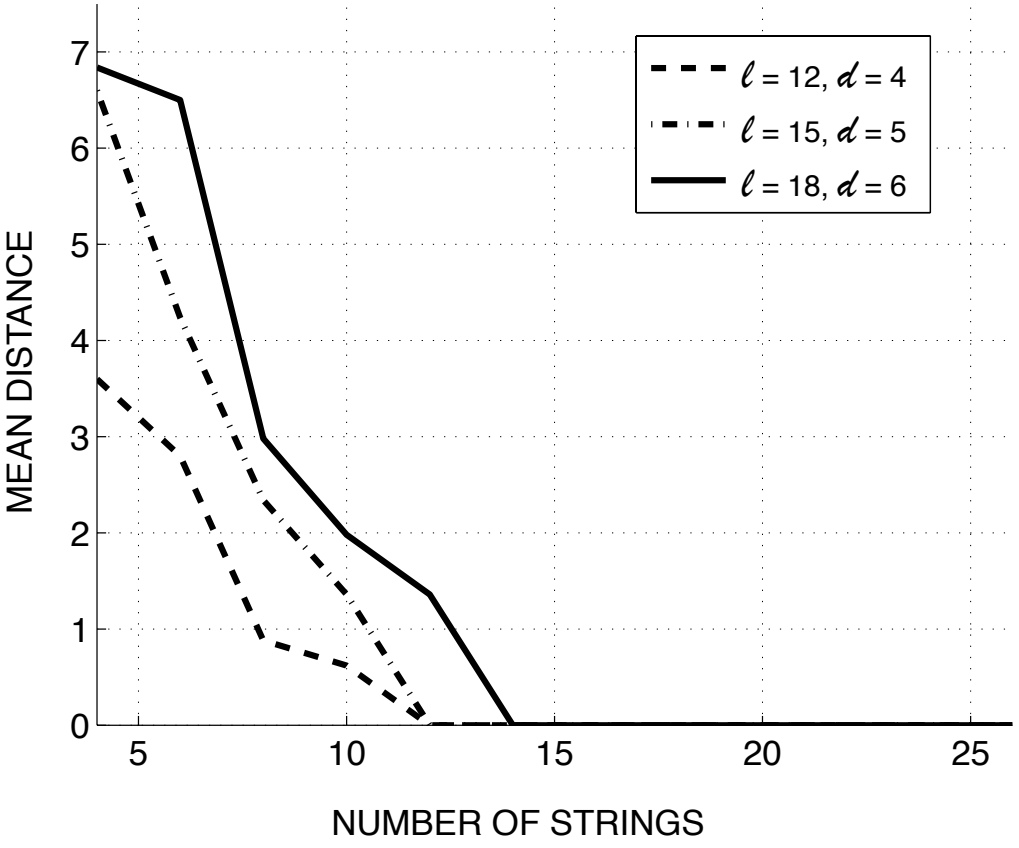

Figure 4.2: An illustration of the decrease in the mean distance between a majority and center string as n increase, for various values of ℓ and d. Each line represents the mean Hamming distance for differing values of n.

4.3.2 Empirical Evaluation of CSP-Greedy

CSP-Greedy begins with a string chosen at random from all majority strings, $s_{maj,0}$, and iteratively augments $s_{maj,0}$ so each time it is closer to at least one of the strings in S. We consider how the number of augmentations of $s_{maj,0}$ required to obtain a center string is affected if $s_{maj,0}$ is initialized to a majority string, rather than a random string.

n	Mean accuracy	Mean number of augmentations	n	Mean accuracy	Mean number of augmentations
4	57	416	16	99	256
6	86	540	18	100	0
8	87	832	20	100	0
10	92	800	22	100	0
12	98	288	24	100	0
14	98	392	26	100	0

Table 4.4: Data illustrating the change in the accuracy and efficiency of *CSP-Greedy* as the value of n increases. We varied the value of n, and fixed ℓ and d to be 10 and 3, respectively. We generated 1000 motif sets for each value of n, determined the number of augmentations of procedure *augment* to obtain a center string from a majority string, and calculated the mean of these 1000 values.

Number of Input Strings

The number of strings has a substantial effect on the running time of *CSP-Greedy*. Table 4.4 outlines the relationship between the value of n and the accuracy and efficiency of *CSP-Greedy*. The mean accuracy of the algorithm is the percentage of motif sets where *CSP-Greedy* finds a center string. We varied the value of n, and fixed ℓ and d to be 10 and 3, respectively. We generated 1000 motif sets for each value of n, determined the number of augmentations of procedure *augment* to obtain a center string from a majority string, and calculated the mean of these 1000 values. For this set of experiments we increased the maximum of augmentations to 1000. Table 4.4 shows that when n becomes significantly large (*i.e.* n is equal to 18) the majority string is a center string.

Figure 4.3 illustrates the change in the running time of algorithm as n increases. Even for significantly large value of ℓ and d (*i.e.* when ℓ was equal to 29 and d was equal to 10), the algorithm ran extremely efficiently. For each set of values of n, ℓ, and d, 1000 motif sets were generated, *CSP-Greedy* was ran on each set, and the mean running time was recorded. The running time decreases significantly as n increased for all values of ℓ and d considered. Out of all the motif sets we considered, a center string was not obtained for 21 of these sets.

Length/Degeneracy Ratio

We consider the change in the accuracy as ℓ and d increase. Again, we generated 1000 random motif sets, and obtained the mean accuracy and number of augmentations from running Procedure *augment* (with the number of augmentations increased to 1000). Table

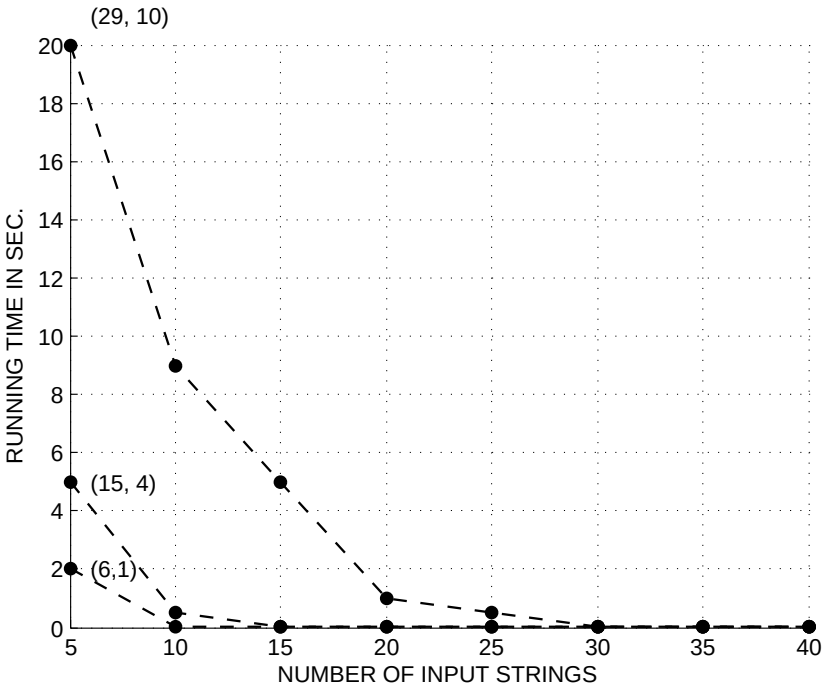

Figure 4.3: An illustration of the time required by *CSP-Greedy* to obtain a center string as n increases, for various values of ℓ and d.

4.5 illustrates the data from these experiments and shows that when $n = 20$ the mean number of augmentations was equal to zero for majority of values of ℓ and d, implying that the majority string was a center string.

		$n = 5$	
ℓ	d	Mean accuracy	Mean number of augmentations
6	1	97	270
8	2	83	272
10	3	67	825
12	3	86	504
14	4	63	813
15	4	69	744
18	6	61	920
21	7	62	912
		$n = 15$	
6	1	100	0
8	2	100	0
10	3	99	225
12	3	100	0
14	4	99	440
15	4	99	506
18	6	96	916
21	7	93	945
		$n = 20$	
6	1	100	0
8	2	100	0
10	3	100	0
12	3	100	0
14	4	100	0
15	4	100	0
18	6	96	183
21	7	97	291

Table 4.5: Data illustrating the change in the accuracy and efficiency of *CSP-Greedy* as ℓ and d increase.The mean accuracy and number of augmentations represents the results obtained for 1000 randomly generated motif sets.

Figure 4.4 illustrates the change in the running time of *CSP-Greedy* as ℓ and d vary, and n is fixed at 20. When the ℓ/d ratio became significantly large the running time was infinitesimal. Again, for each set of values of n, ℓ, and d, 1000 motif sets were generated, *CSP-Greedy* was run on each set, and the mean running time was recorded. Out of the 24,000 sets considered, a center string was not obtained for 42 of the sets.

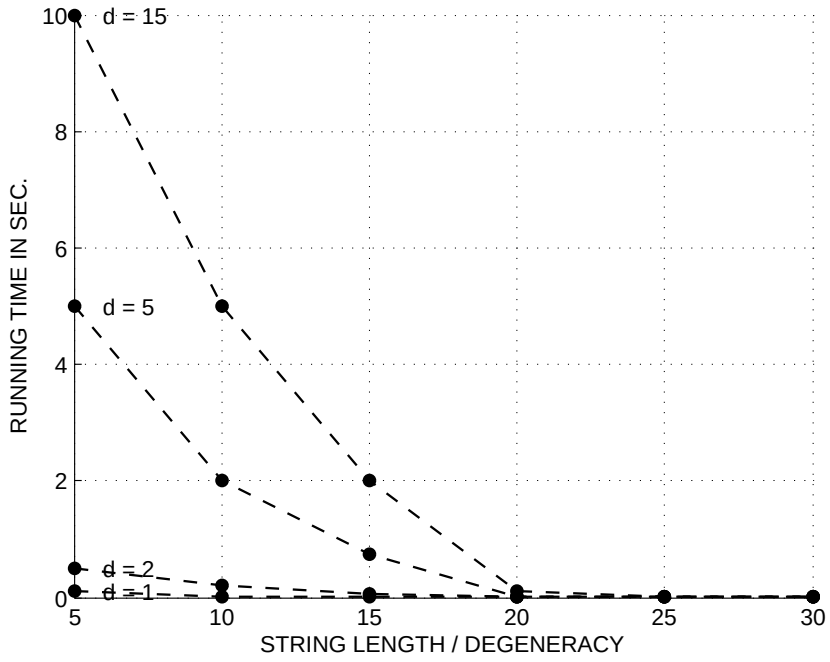

Figure 4.4: An illustration of the time required by *CSP-Greedy* to obtain a center string as ℓ/d ratio increases, for various values of d. In these experiments, n is equal to 20.

4.3.3 Application to Motif Recognition

MCL-WMR was developed specifically for the problem of detecting weak motifs in genetic data and works by first building a weighted graph model of the given motif-recognition problem and then using a graph clustering algorithm to quickly determine important subgraphs that need to be searched further for valid motifs. These smaller subproblems are then solved optimality using a dynamic-programming algorithm for finding motifs in dense subgraphs. We extend MCL-WMR to incorporate *CSP-Greedy*. This new algorithm, *pMCL-WMR*, detects motifs in data sets with a large number of strings. More specifically, pMCL-WMR efficiently discovers motifs in data sets that have 30 or more strings, and finds regulatory strings in genomic data.

4.3.4 Performance of pMCL-WMR on Synthetic Data

We produced problem instances as follows: we chose a random center string of length ℓ, and picked m occurrences of the motif by randomly choosing d positions per occurrence and randomly mutating the base at each. We constructed m background strings of length n and inserted the generated motifs into a random position in the string. For each of the (ℓ, d) combinations, 100 random sets of input strings ($n = 20$, $m = 1000$) were generated. The running time is given in CPU seconds.

Table 4.6 shows the comparison between the running time of MCL-WMR and that of pMCL-WMR. Two significant trends are witnessed in the data: pMCL-WMR is capable of solving hard instances of motif recognition (*i.e.* when $\ell = 25$ and $d = 7$) and pMCL-WMR gives a dramatic improvement over MCL-WMR with respect to the running time for all values of ℓ and d. The main advantage to our tool is the time required to solve the extremely difficult challenge problems – from the $(18, 6)$ to the $(25, 7)$ problem – in substantially better running time and with 100% accuracy.

The computational results in Subsection 4.3.1 inspire the investigation of instances with a significantly large number of sequences – that is, instances where n varies from values greater than 20. Table 4.7 shows the evaluation of the performance of pMCL-WMR on a range of problems with an increasing number of sequences. The efficiency on these sets of problems is noteworthy ranging from 781 (when $n = 18$) to 2781 ($n = 30$). As far as we are aware, these are the first computational experiments where n is larger than 20.

An entry "-" indicates that the program was unable to solve the specific problem in a reasonable amount of time. The mean running time is given, followed by the standard deviation.

(ℓ, d)	MCL-WMR	pMCL-WMR
(10, 2)	1003 / 32	518 / 11
(15, 4)	3232 / 92	788 / 22
(16, 5)	4200 / 103	1529 / 31
(18, 6)	8905 / 150	1723 / 47
(25, 7)	-	2923 / 45

Table 4.6: Comparison between the performance of MCL-WMR and that of pMCL-WMR on synthetic data. In all experiments, $m = 1000$, $n = 20$, and ℓ and d are varied. The mean and standard deviation of the running time in CPU seconds is given.

n	MCL-WMR	pMCL-WMR
18	1981 / 65	781 / 23
20	3233 / 82	931 / 45
24	-	1721 / 72
28	-	1200 / 87
30	-	2781 / 91

Table 4.7: Comparison between the performance of MCL-WMR and that of pMCL-WMR on synthetic data as n increases. In all experiments, $\ell = 15$, $d = 4$ and $m = 1000$. The mean and standard deviation of the running time in CPU seconds is given.

4.3.5 Using pMCL-WMR to Find Regulatory Elements

An important biological challenge is to determine regulatory elements in DNA – specifically, binding sites for transcription factors. In this section, we demonstrate the use of pMCL-WMR in discovering these DNA sequence "motifs" in data sets with a large number of DNA sequences. Tompa *et al.* [109] extensively assess 13 motif-recognition tools using test sets that make use of transcription factor binding sites. The binding sites were obtained from the TRANSFAC database [117] and contain eukaryotic transcription factors. The TRANSFAC database is extremely comprehensive, containing data from a large variety of species (*i.e.* species include yeast, *mus*, *oryctolagus cuniculus*, and *homo sapiens*) [117]. For more details concerning the data set, including the selection process for transcription factors and binding sites from TRANSFAC, see Tompa *et al.* [109].

Each transcription factor gives rise to one set of sequences. The number of sequences varied from 34 to 6 and the sequence length (parameter m) varied from 700bp to 2000bp. The transcription factor binding sites vary in length and thus, in order to assess pMCL-WMR, we ran the program on varied values ℓ and d. The lengths of the motifs were same as those of the published motifs and d varied. Experimental results are shown in Table 4.8. pMCL-WMR was capable of discovered all of the motifs sets reported by Tompa *et. al.* [109], as well as some undetected motifs. The motifs discovered only by pMCL-WMR and not by any of the motif-recognition programs assessed by Tompa *et. al.* [109] are shown in Table 4.8.

4.4 Summary and Open Problems

In this chapter, we gave a linear-time algorithm for solving Closest String problem with four binary strings, proved lower and upper bounds for the approximation guarantees for a simple probabilistic algorithm for the general problem, and showed the existence of an

Data set	(ℓ, d)	n	Motif pattern discovered
hm06	(7, 1)	9	TttTcccC
hm06	(8, 1)	9	GGAcTGCT
hm10	(8, 3)	6	TTTcgCGC
hm13	(10, 2)	6	aaAATTatTC
hm13	(11, 1)	6	tccCcCACAaa
hm19	(11, 2)	5	tccCcCACAaa

Table 4.8: Illustration of the regulatory strings found by pMCL-WMR. Data are collected from SCPD. For each set of data, we determined motifs of the same length using pMCL-WMR, and gave the motif published by Tompa *et. al.* [109] for the same data set.

Data set	(ℓ, d)	n	Time
hm06	(7, 1)	9	80
hm06	(8, 1)	9	102
hm10	(8, 3)	6	210
hm13	(10, 2)	6	178
hm13	(11, 1)	6	220
hm19	(11, 2)	5	321

Table 4.9: CPU time required by pMCL-WMR to detect the regulatory sequence patterns shown in Table 4.8.

algorithm with running time $O(\ell^3)$ that achieves a $\left(1 - \left(\frac{\epsilon}{2^n}\right)^\ell\right)$-approximation, for some small $\epsilon > 0$ and significantly large values of n. Our smoothed analysis of *CSP-Greedy* gives reasons as to why this algorithm and the $O(n\ell + nd \cdot d^d)$-time algorithm of Gramm *et al.* [57] is efficient in practice for Closest String instances with a large number of strings. This analysis gives an analytical reason to why this – and perhaps other similar Closest String algorithms – perform well in practice. Lastly, we gave empirical results that demonstrate the majority string is likely to be a center string when the number of sequences is moderately large.

There exist numerous open problems that warrant further investigation, including the following:

- Since the publication of the linear-time algorithm presented in this chapter, these results have been extended by Amir *et al.* [3] to a variant of the Closest String problem. Determining how these results could be generalized to larger alphabets or to instances containing more than four strings remains open. Such a generaliza-

tion would invite many open problems in motif recognition to be revisited, as their tractability might be determined more concretely.

- In smoothed analysis one often analyzes how fragile worst case instances are. Manthey and Reischuk [80] suggest examining the dual property of how robust the best (or good) instances are. Lastly, we propose continuing this form of analysis and determining how stable the best-case instances of the closest string problem are under perturbations. This would involve considering instances where *CSP-Greedy* performs efficiently and showing that even large perturbations of these instances yield problems that can be solved in efficient time.

- There exist several open problems related to the smoothed complexity of string selection problems that warrant investigation. Analyzing the smoothed complexity of the PTAS of Li *et al.* [70] for the Closest String problem requires further study. The running time of this PTAS is detrimentally large, thus reducing the algorithm to only theoretical importance. However, there have been several papers attempting to prove that the algorithm performs significantly better in practice [22, 23]; the smoothed analysis of the algorithm would potentially complete this area of study.

Chapter 5

Closest String with Outliers

Finding similar regions in multiple DNA, RNA, or protein sequences plays an important role in many applications, including universal PCR primer design [39, 67, 74, 91], genetic probe design [67], antisense drug design [36, 67], finding transcription-factor binding sites in genomic data [109], determining an unbiased consensus of a protein family [10], and motif recognition [67, 86, 88]. Up to this point, we have formulated these problems with respect to the Closest String problem, which was first introduced and studied in the context bioinformatics by Lanctot *et al.* [67]. Since its introduction, the investigation of efficient polynomial time approximation algorithms and exact exponential time algorithms for the Closest String problem has been thoroughly considered [45, 46, 57, 67, 70, 75, 77]. However, in many contexts the Closest String problem can be too restrictive. In this chapter we consider the computational tractability and intractability of a less-restrictive version of this problem, and in the following one, we illustrate the applicability of this version.

The Closest String problem requires that the Hamming distance constraint be satisfied for each of the input strings and therefore, is robust to the overrepresentation of the input strings; regardless of the number of occurrences of a distinct string, the Hamming distance constract must be still satisfied. For this reason it is frequently used to model many of the aforementioned applications. However, this property also causes a severe problem: if the input includes a string that is significantly different from the other input strings, which we refer to as an "outlier", then it will have the effect that a centre string for the complete set of input strings will not exist; d will have to be increased dramatically to account for this string and obtain a center string. This is a significant limitation for applications such as the design of universal primers where a small d is crucial for the effectiveness of the primers. In this and many other applications, it would be preferable to determine a "good" center string (*i.e.* one that is reasonably close to each of the strings) for a large portion of the input strings rather than trying to find a center string

for the complete set and in doing so finding one that is a large distance from many or all of the strings. Hence, we aim to model the task of finding a center string that is within (reasonably small) distance d of most of the input strings, not necessarily all. Another compelling consequence of the modification of the model is that in situations where a more satisfying solution can be found by regarding a few strings as outliers, the initial decision of including them requires re-examination.

We formally model this problem as follows:

Closest String with Outliers (CSWO)
Input: A set of n length-ℓ strings $S = \{s_1, \ldots, s_n\}$ over a finite alphabet Σ and nonnegative integers k and d.
Question: Find a center string s and a subset of $S^* \subset S$, such that $|S^*| = n - k$ and $d(s,t) \leq d$ for $t \in S^*$.

We denote $n - k$ as n^*, and the symbol at position p of string s_i to be $s_i(p)$.

There exists a simple reduction from the Closest String problem to CSWO that demonstrates it is NP-complete even in the special case where the alphabet is binary and $k = 0$, implying it is unlikely to be solved exactly by a polynomial-time algorithm, unless P=NP. One approach to investigating the computational intractability of CSWO is to consider its parameterized complexity, which aims to classify computationally hard problems according to their inherent difficulty with respect to multiple parameters of the input. If it is solvable by an algorithm that is polynomial in the input size and exponential in parameters that are typically small then it can still be considered tractable in some practical sense.

Smith [102] introduced a related optimization problem. Given a set $S = \{S_1, \ldots, S_n\}$ of m-length sequences over the alphabet Σ and integers d and ℓ, the aim of the Maximum Coverage Approximate Substring problem is to maximize $|S'|$, $S' \subseteq S$, such that for some $s \in \Sigma^\ell$ and for all $S_i \in S$, where there exists a substring $s_i \in S_i$ such that $d(s, s_i) \leq d$. Smith demonstrated that this problem is APX-hard, and gives a $|\Sigma|^d$-approximation algorithm for this problem.

For unbounded alphabet size, we show that CSWO is W[1]-hard for every combination of the parameters ℓ, d, and n^* and thus, is fixed parameter intractable when parameterized by any subset of these parameters, unless FPT = W[1]. We also show that when the alphabet is unbounded, there exists a fixed-parameter tractable algorithm for CSWO with respect to the parameters d and k. In the case of a constant-size alphabet CSWO is fixed parameter tractable for the parameter n but intractable for the parameter k. The complexity of the problem remains open when parameterized by d and the alphabet is of constant size, and when parameterized by n^* and k.

5.1 Closest String with Outliers: Tractability Results

We first consider Σ as a parameter. In computational biology applications the biological sequences of interest are typically DNA or protein sequences and hence the number of different symbols is a small constant (*i.e.* 4 or 20 in the case of DNA or protein sequences, respectively). Restricting Σ only does not make CSWO tractable since it is NP-hard even when the alphabet is binary. However, if Σ and ℓ are both parameters then it is fixed-parameter tractable; we can enumerate and check all the $|\Sigma|^\ell$ possible center strings. We show in this section that the problem is fixed parameter tractable with respect to the parameters Σ, ℓ, d and n. We will prove in a later section that it is imperative that Σ be a parameter in order to obtain this tractability.

Algorithm 6 CSWO Algorithm

Input: A CSWO instance with a set of S n strings of length ℓ, parameters Δd, d and k, and a candidate string x.
Output: A string s^* if there exists a set S of at least n^* strings where each string in S has distance at most d from s^*, and "Not found" otherwise.

If $\Delta d < 0$ or $k < 0$ then return "Not found"
Choose $i \in \{1, \ldots, n\}$ such that $d(x, s_i) > d$. If no such i exists return x.
s_{ret} = CSWO Algorithm ($S \setminus \{s_i\}$, Δd, $k-1$, x)
If s_{ret} = "not found " then
    $\mathcal{P} = \{p \mid x(p) \neq s_i(p)\}$;
    Choose any $\mathcal{P}'$ from $\mathcal{P}$ with $|\mathcal{P}'| = d+1$.
    For each position $p \in \mathcal{P}'$
        Let x be equal to s_i at position p
        s_{ret} = CSWO Algorithm (S, $\Delta d - 1$, k, x)
        If $s_{ret} \neq$ "not found ", then return s_{ret}
Return "not found"

The fixed parameter algorithm that we present is very similar to the algorithm presented by Gramm *et al.* [57], where it is proved that Closest String is fixed parameter tractable with respect to the parameter d. In the algorithm by Gramm *et al.* [57] at each recursive step a string s is selected that has Hamming distance at least $d+1$ away from the current candidate center string x if one exists; otherwise x is returned since it is a center string. Then for any $d+1$ positions where x and s disagree, there is at least one position at which s is equal to the final solution. The algorithm tries each of the $d+1$ positions, changes x to s at one of the $d+1$ positions, reduces Δd by one, and calls itself recursively. Since the recursion stops after at most d steps, the size of the search tree is bounded by $O((d+1)^d)$.

Our algorithm begins with s_1 as the candidate center string. If s_1 is a center string with respect to S then we are done; otherwise there exists a string s_i that has distance at least $d+1$ from s_1. We determine whether s_i belongs in the set of outliers by trying both possibilities: s_i belonging in the set of outliers and s_i not belonging in the set of outliers. If it is an outlier then we remove it from S and recurs on the smaller set with $k-1$. If it is not an outlier then we use s_i to move the candidate string x closer to toward s_i, which can be done by applying the methodology of Gramm *et al.* [57]. This will increase the size of the search tree.

Proposition 5.1.1 *The* CSWO Algorithm *solves the* CSWO *problem in time* $O(n\ell + nd \cdot d^d \cdot 2^{k+d})$.

Proof We first consider the running time of the algorithm and then subsequently, give proof of the algorithm's correctness.

Running time. Each recursion of the algorithm reduces either k or d by 1. Thus, there are at most $k+d$ guesses of whether a particular string belongs in the set of outliers. Thus, the search tree size is increased by a multiplicative factor of at most 2^{k+d} and the search tree size is bounded above by $O(2^{k+d} \cdot (d+1)^d)$. The analysis of Gramm *et al.* [57] demonstrated that each recursive step takes time $O(nd)$ and the preprocessing time takes $O(n\ell)$ and therefore, we obtain an overall running time of $O(n\ell + nd \cdot d^d \cdot 2^{k+d})$.

Correctness We show the correctness of our algorithm by showing the correctness of the first recursive step and then the correctness of the algorithm follows by inductively applying the following argument. Clearly, if S does not contain a subset S^* of n^* strings, such that there exists a center string s^* for S^* then "not found" will be returned and therefore, we assume otherwise.

If s_1 is a center string for S then the algorithm immediately halts so we assume there exists a string s_i in S that does not have s_1 as a center string. When considering s_i, there are two subcases: one where s_i is in the set of outliers, and another where s_i is not. Suppose s_i is in the set of outliers; then the first case will successfully remove s_i from the set and recurse on $S \setminus \{s_i\}$. Otherwise, if s_i is not in the set of outliers, then eventually the second case will be reached. We refer to the set of positions as *correct* if $\{p \mid s_1(p) \neq s^*(p) = s(p)\}$. It follows from Gramm *et al.* [57] that one of the $d+1$ chosen positions p will be a correct one. Thus, we have shown that either one of the subcases will lead to a smaller subcase containing the solution for S. □

The previous result demonstrates the fixed-parameter tractability with respect to d and k. We note that a similar modification of the $O(n|\Sigma|^{O(d)})$ algorithm of Ma and Sun [77] also gives a fixed parameter algorithm with respect to the parameters Σ, d and k. In the

modified algorithm, for any string s with distance greater than d to the current candidate center string x, we again try the subcases where s is an outlier, and is not an outlier. In the former case, we remove s from the set of input strings S and recurs on S and $k-1$, and in the latter case, we use the same technique as in the algorithm of Ma and Sun [77] to reduce the distance between x and the final solution. This modification that accounts for the outliers results an extra multiplicative factor of $O(2^{k+\log d})$ to the running time of the original algorithm. Although this algorithm improves upon the running time of the previous result, it requires that Σ is also a parameter. Further, we note that some of the recent improvements [28, 115, 119] to the algorithm of Ma and Sun can be modified in a similar manner to obtain fixed parameter algorithms for CSWO with respect to parameters Σ, d and k.

Proposition 5.1.2 CSWO *is fixed parameter tractable for parameters* Σ *and* n.

Proof Gramm *et al.* [57] gave a fixed-parameter tractable algorithm for Closest String with respect to the number of strings and Σ, which we refer to this algorithm as *ILP-procedure(S)*, where S is the set of input strings. Our algorithm enumerates all size-n^* subsets of S, and calls *ILP-procedure* on each subset. □

The algorithm proposed in the proof of Proposition 5.1.2 has $O\left((n \cdot f(\ell, n, d))^n\right)$ running time, where $f(\ell, n, d)$ is the time required for the fixed parameter tractable algorithm. The algorithm of Gramm *et al.* [57] models the problem as an integer linear program with n^{n+1} variables and constraints and then applies the famous result by Lenstra [69], which states that an integer linear program can be solved in polynomial time with a constant number of variables. Specially, the result is as follows:

Theorem 5.1.3 *[69] The integer programming feasibility problem can be solved in* $O(p^{9p/2} \cdot L)$ *time, where* p *is the number of variables and* L *is the number of bits in the input.*

Proposition 5.1.2, as well as the original result of Gramm *et al.* [57], is only or theoretical use since the combinatorial explosition in n in the running time of the corresponding algorithms is huge, thus rendering these algorithms impractical.

5.2 Closest String with Outliers: Intractability Results

We derive the W[1]-hardness result by a series of intermediate steps, aiming at a reduction from Clique to Closest String with Outliers. This shows that Closest String with Outliers is W[1]-hard for the combination of ℓ, d, and n^*, when the alphabet is unbounded.

5.2.1 Reduction from CLIQUE

As previously described, we let the CLIQUE instance be given by an undirected graph $G = (V, E)$ with a set $V = \{v_1, v_2, \ldots, v_n\}$ of n vertices, a set E of m edges, and a positive integer t denoting the size of the desired clique. We describe how to generate a set S of $\binom{t}{2}|E|$ strings such that G has a clique of size t if and only if there is a subset of S of size $\binom{t}{2}$, denoted as S^*, where there exists a string x such that $d(s_i, x) \leq d$ for all $s_i \in S^*$. We let $\ell = t$ and $d = t - 2$. We assume that $t > 2$ since $t \leq 1$ produces trivial cases.

We begin by describing the alphabet. We assume $|\Sigma|$ can be unbounded, however, for any given instance obtained by our reduction from CLIQUE, $|\Sigma|$ is finite. We define Σ to be equal to the union of the following sets of symbols:

1. $\{v_i|$ for all $i = 1, \ldots, |V|\}$. There exists one symbol representing each vertex in G.
2. Let $m = |E|$ then $\{c_{i,j,m}|i = 1, \ldots, t;\ j = 1, \ldots, t\}$. There exists an unique symbol for each of the $\binom{t}{2} \cdot |E|$ strings produced for our reduction.

Hence, we have a total of $|V| + \binom{t}{2} \cdot |E|$ symbols.

Next, we generate a set of $\binom{t}{2}|E|$ strings $S = \{s_{1,1,1}, \ldots, s_{1,1,|E|}, s_{1,2,1}, \ldots, s_{1,2,|E|}, \ldots, s_{t-1,t,|E|}\}$. Every string has length t and will encode one edge of the input graph. For string $s_{i,j,m}$ we encode edge $e_m = (v_r, v_s)$, where $1 \leq r < s \leq |V|$, but by letting position i equal to v_r and position j equal to v_s and the remaining positions equal to $c_{i,j,m}$. Hence, a string is given by

$$s_{i,j,m} := [c_{i,j,m}]^{i-1} v_r [c_{i,j,m}]^{j-i-1} v_s [c_{i,j,m}]^{m-j}.$$

Clearly, this reduction runs in time $O(|V| + \binom{t}{2} \cdot |E|)$

To clarify our reduction, we give an example. Let $G = (V, E)$ be an undirected graph with $V = v_1, v_2, v_3, v_4$ and edges $E = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3)\}$ and let our CLIQUE instance have G and $t = 3$. Figure 5.1 illustrates the reduction. Using G, we exhibit the above construction of $\binom{t}{2} \cdot |E| = 12$ strings, which we denote as S. We claim that there exists a clique of size 3 if and only if there exists a string s^* of length $\ell = t = 3$ and subset S^* of S of size 3 where $d(s, s_i) \leq d$ for all $s_i \in S^*$. In this example, the center string s is equal to $v_1 v_2 v_3$ and each string in the set $\{v_1 v_2 c_{121}, v_1 c_{132} v_3, c_{234} v_2 v_3\}$ is such that each string in S^* has Hamming distance at most 1 from s.

5.2.2 Correctness of the Reduction

The following two lemmas establish the correctness of the reduction.

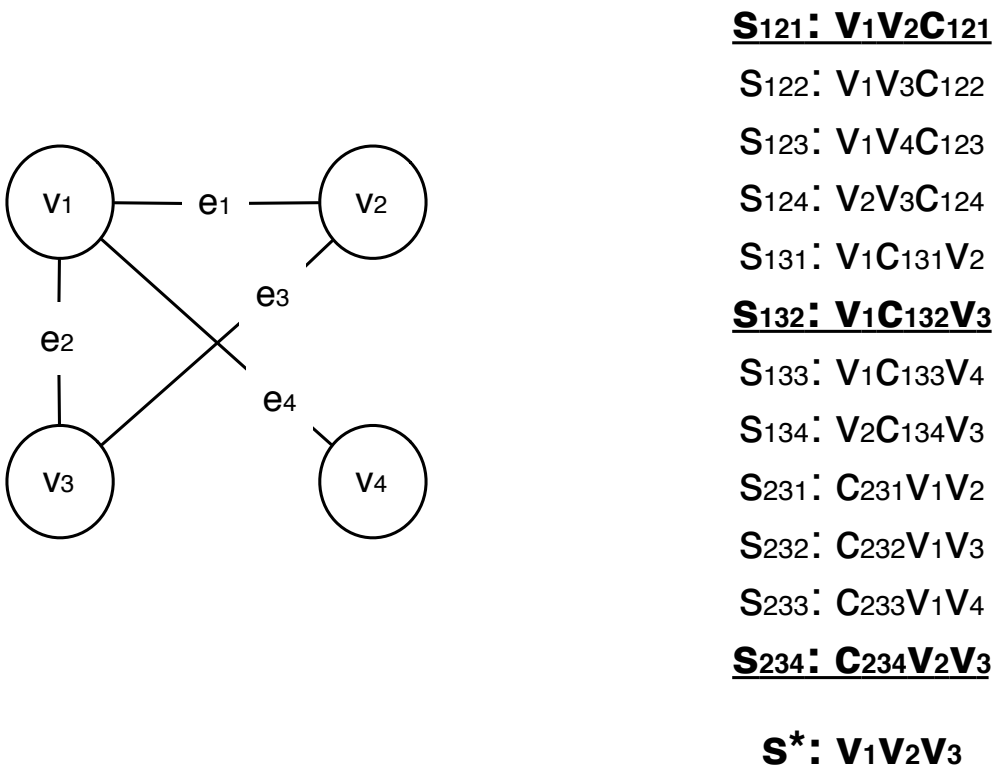

Figure 5.1: An example of the parameterized reduction from a CLIQUE instance G with $t = 3$ to an instance of CSWO with 12 strings with $\ell = t = 3$, $d = t - 2 = 1$, and $n^* = \binom{3}{2} = 3$. In bold we have the set of strings $S^* = \{s_{121}, s_{132}, s_{234}\}$ that corresponds to the clique containing the vertices $\{v_1, v_2, v_3\}$. We note that S^* is the only set of size 3 of strings with Hamming distance at most d from the center string s^*.

Lemma 5.2.1 *For a graph with a t-clique, the construction in Subsection 5.2.1 produces a* CSWO *instance with a set S^* and a string s of length ℓ such that for every $s_i \in S^*$ $d(s_i, s) \leq d$.*

Proof Let the input graph have a clique of size t. Let $v_{\alpha_1}, v_{\alpha_2}, \ldots, v_{\alpha_t}$ be the vertices in the clique C of size t and without loss of generality, assume $\alpha_1 < \alpha_2 < \ldots < \alpha_t$. Then we claim that there exists a subset of $\binom{t}{2}$ vertices that have distance at most $t - 2$ from the string $s = v_{\alpha_1} v_{\alpha_2} \ldots v_{\alpha_t}$. Consider the first edge of the clique $(v_{\alpha_1}, v_{\alpha_2})$. The string $s_{11r} = v_{\alpha_1} v_{\alpha_2} [c_{11r}]^{t-2}$, where edge r has endpoints v_{α_1} v_{α_2}, is contained in the set of strings $\{s_{111}, s_{112}, \ldots, s_{11|E|}\}$. Clearly, $H(s_{11r}, s) = t - 2$. For each edge in C we have we have a

string in S that has distance at most $t-2$ from s, as required. □

For the reverse direction, we need to prove that the existence of a subset S^* of $\binom{t}{2}$ and a string s where $d(s, s_i) \leq t-2$ for all $s_i \in S^*$ implies the existence of a clique in G with t vertices.

Lemma 5.2.2 *If (S, ℓ, d, n, k) is a 'YES' instance of the* CSWO *problem then (G, t) is a 'YES' instance of the* Clique *problem.*

Proof Hence, we are required to show that the t symbols of the center string correspond to the t vertices of clique in the input graph. Let S^* be the subset of S of size $\binom{t}{2}$ such that s has distance $t-2$ from each string in S^*. Since $\ell = t$, $n^* = t$, $d = t-2$ and for each symbol $c_{i,j,m}$ there exists only a single string $i = 1, \ldots, t$, $j = 1, \ldots, t$ and $m = 1, \ldots, |E|$ it follows from the Pigeonhole principle that the center string s only contains symbols from $\{v_i|$ for all $i = 1, \ldots, |V|\}$. Without loss of generality assume s is equal to $v_{\alpha_1} v_{\alpha_2} \ldots v_{\alpha_t}$ for $\alpha_{v_1}, \alpha_{v_2}, \ldots, \alpha_{v_t} \in \{1, \ldots, |V|\}$. Consider any pair α_i, α_j for $1 \leq i < j \leq t$ and consider the set of strings $S_{i,j} = \{s_{i,j,1}, s_{i,j,2}, \ldots, s_{i,j,|E|}\}$. Recall that $S_{i,j}$ contains a string corresponding to each edge $e = (r, s)$ in E which has v_r at the ith position and v_s at the jth position and $c_{i,j,m}$ at all remaining positions. Therefore, we can only find a string in $S_{i,j}$ that has distance at most $t-2$ from s if v_{α_i} is at the ith position and v_{α_j} is at the jth position; and such a string exists if and only if there is an edge in G connecting v_{α_i} to v_{α_j}. Hence, the center string s implies there exists an edge between any pair of vertices in G in the set $\{v_{\alpha_1} v_{\alpha_2} \ldots v_{\alpha_t}\}$ and by definition the vertices form a clique. □

Our main theorem follows directly from Lemma 5.2.1 and Lemma 5.2.2. We note that the hardness for the combination of all three parameters also implies the hardness for each subset of the three.

Theorem 5.2.3 CSWO *with unbounded alphabet is W[1]-hard with respect to the parameters ℓ, d, and n^*.*

Since there exists a trivial reduction from the Closest String problem to CSWO (*i.e.* simply set $k = 0$ in CSWO), there cannot exist a fixed parameter tractable algorithm for CSWO with k as a parameter, unless P = NP; such an algorithm would contradict the NP-hardness of Closest String.

Fact 3 CSWO *is W[1]-hard with respect to the parameter k, for any fixed $|\Sigma| \geq 2$.*

Parameter(s)	$\|\Sigma\|$ is a parameter	$\|\Sigma\|$ is unbounded
ℓ, d, n^*	FPT (trivial)	W[1]-hard (*)
ℓ	FPT (trivial)	W[1]-hard (*)
d, n^*	Open	W[1]-hard (*)
d, k	FPT (*)	FPT (*)
n^*, k	FPT	Open
k	W[1]-hard (trivial)	W[1]-hard (trivial)

Table 5.1: An overview of the fixed parameter tractability and intractability of the CSWO. Asterisk denotes the new results discussed in this chapter.

5.3 Summary and Open Problems

We introduced the CSWO problem, and proved that with unbounded alphabet size and parameterized by ℓ, d and n^* it is W[1]-hard. We also gave fixed parameter algorithms for the problem when parameterized by d and k, and with unbounded alphabet size. In the case of a fixed alphabet size, we showed CSWO is fixed parameter tractable when parameterized by $n = n^* + k$. Table 5.3 summarizes these tractability and intractability results. Currently, the fixed parameter tractability of the CSWO problem when parameterized by d, n^* and Σ, and by n^* and k, remains open (see Table 5.3).

In addition, the existence of efficient, non-trivial approximation algorithms for this problem warrants further investigation. Smith [102] proved Maximum Coverage Approximation Substring is APX-hard, however, the reduction does not directly extend to the specific case where $m = \ell$ (*i.e.* CSWO). Therefore, it currently remains open as to whether CSWO is APX-hard. He also proved that the $|\Sigma|^d$-approximation algorithm for Maximum Coverage Approximation Substring provides a $(d+1)$-relaxed decision procedure for CSWO. A ρ-decision procedure is an optimization algorithm that finds a solution with performance ratio ρ, or correctly concludes that no exact solution exists.

There are many open problems concerning the approximability of CSWO and Maximum Coverage Approximation Substring. As mentioned by Smith "Maximum Coverage Approximation Substring has a large margin for improvement in the complexity bounds" [102, page 80]. Among the problems outlined in this chapter, one of the key open problems is the development of a polynomial algorithm that gives a constant approximation guarantee, even for alphabets of constant size.

Chapter 6

Fast Motif Recognition via Statistical Thresholds

MCL-WMR uses graph clustering to determine pairwise bounded sets that might be valid motifs. The major impediment to the efficiency of MCL-WMR was the exponential-time refinement algorithm used to determine which "candidate motif sets" (*i.e.* pairwise bounded sets) have a center string; this step becomes a bottleneck for solving challenging weak motif instances, such as $(18, 6)$, when the number of such candidate sets increases dramatically [24]. In Chapter 4, we give a probabilistic heuristic for solving the Closest String problem, which filters candidate sets based on a majority vote string, that has acceptable accuracy when n is significantly large (*i.e.* when $n \geq 20$). In this chapter, we propose a probabilistic algorithm that eliminates the need for a strong bound on n; our algorithm uses the weight of the set to determine quickly and with a small probability of error whether the set is a decoy set or a motif set.

We defined the weight of a set of strings S as the sum of the pairwise Hamming distances, *i.e.* $\sum_{1 \leq i < j \leq n} H(s_i, s_j)$. If the weight of a set, which can be calculated in polynomial time, can be used to indicate whether it is a motif set or a decoy set then the Closest String string can be solved extremely efficiently and accurately in practice – simply calculate the weight of the pairwise bounded set and decide whether the set has a center string based on this value. For this heuristic to work we need to know how the respective weights of a random motif set and a random decoy set are distributed. Further, the distributions need to be adequately separated so that the weight of a set leaves little ambiguity as to whether the set is a motif set or a decoy set.

There exists an algorithm to sample from the set of all motif sets: simply choose any length-ℓ string as the center string and sample with replacement from the set of all strings that are at distance at most d from that sequence. This sampling algorithm and its appropriateness was discussed in Chapter 3. Unfortunately we do not know an analogous

sampling algorithm, either exact or approximate, for decoy sets. If we could sample pairwise bounded sets uniformly at random, then we could learn the probability distribution of the weight of a random decoy set.

We give a method to generate pairwise bounded sets uniformly, use this method to determine the probability distribution of the weight of a random decoy set, and show the existence of a separation between this distribution and the probability distribution of the weight of a random motif set. Thus, we solve Closest String instances accurately and efficiently using the simple heuristic of using the weight as an indicator as to whether a pairwise bounded set is a motif set or a decoy set. The separation of the distributions becomes increasingly more prevalent as the number of strings in the set (*i.e.* the parameter n) increases, so the accuracy of our method increases as the number of strings increases.

We significantly extend our earlier motif-recognition program, MCL-WMR, by incorporating the heuristic for the Closest String problem described in this chapter. This new algorithm, *sMCL-WMR*, detects motifs in data sets with a large number of strings (*i.e.* 30 or more strings). sMCL-WMR represents the input data as a weighted graph and uses graph clustering to narrow the search to smaller problems that can be solved with significantly less computation. An efficient refinement algorithm that distinguishes valid motif sets from decoy sets allows sMCL-WMR to detect motifs in very large data sets in significantly less computational time than MCL-WMR.

Finally, we illustrate the applicability of sMCL-WMR and another variant, MCL-FSP, in analyzing the genomic data of canola. Using these programs, we identify more than 40 motifs in promoters conjectured to be responsible for seed coat-specificity. Based on these motifs, a promoter DNA sequence of approximately 700 bp was synthesized and introduced into canola with the aim of obtaining its biological expression.

6.1 Sampling Pairwise Bounded Sets

We discuss uniform sampling, or generation, of pairwise bounded sets. A standard method used to generate a random motif set is to choose an length-ℓ string uniformly at random from all possible 4^ℓ strings to be the center string, and then form a motif set by selecting n strings at random with replacement from the set of all strings with Hamming distance at most d from this center string [16, 24]. This method samples a motif set uniformly at random, and further corresponds to how synthetic problem data sets are constructed. For example, synthetic problem instances are traditionally generated as follows: a random center string of length ℓ is chosen, n occurrences of the motif are generated by randomly mutating at most d positions, and each of the n motif instances is embedded at a random location into a different background string of length m. We note that other non-uniform distributions have also been used to generate motif sets [88].

When sampling uniformly from a poorly understood sample set, *rejection sampling* is a naïve but useful technique. If we can find a superset of the target set that is easy to sample from uniformly, we can sample from this superset and simply throw away (reject) any sampled element that is not in the target set. To sample uniformly at random from all pairwise bounded sets using rejection sampling in the most naïve way, we would generate n random length-ℓ strings and accept the set if it is pairwise bounded, and reject and repeat otherwise. However, since it is unlikely that such a randomly generated set would be pairwise bounded, this method is extremely inefficient. We introduce a heuristic to generate random sets that are more likely to be pairwise bounded, thus speeding up the rejection sampling process enough to be practical.

We generate the first string, s_1, uniformly at random from the set of all length-ℓ strings then generate each of $s_2, \ldots, s_n$ in turn uniformly at random from the set of all strings at distance at most $2d$ from s_1. This gives us a set of strings generated uniformly at random from the set of all strings that have s_1 as the first string and each other string at distance at most $2d$ from s_1. If the set is pairwise bounded we keep it; if it is not we reject it and start over. The fact that this method generates pairwise bounded sequences uniformly at random can be verified by induction on n.

The number of times a set of n strings is considered and rejected until a pairwise bounded set is generated follows a geometric distribution and therefore, the efficiency of this method is determined by the probability that a set is rejected. Though this method is fast enough to work in practice for values of n we are interested in, the expected running time when generating a single pairwise bounded set grows exponentially with n.

Definition (Neighbourhood) For a string $s \in \Sigma^\ell$, the (ℓ, k)-neighbourhood of s, denoted as $N_{\ell,k}$, is the set

$$\{s_i | s_i \in \Sigma^\ell, d(s_i, s) \leq k\}.$$

We note that $|N_{\ell,k}| = \sum_{i=0}^{k} \binom{\ell}{k} (|\Sigma| - 1)^k$.

Proposition 6.1.1 *The probability that a set generated using the above method is pairwise bounded decreases at least exponentially fast as a function of n.*

Proof For $1 \leq i \leq n$ let S_i be the subset of S containing the first i randomly chosen strings, with $S_n = S$. Let A_i be the event that S_i is pairwise bounded. Any subset of a pairwise bounded set is pairwise bounded, so A_i implies A_{i-1} for $2 \leq i \leq n$. Therefore by Bayes' law we have $\Pr(A_i) = \Pr(A_i | A_{i-1}) \Pr(A_{i-1})$. To prove that $\Pr(A_n)$ decays exponentially with n we need only show that $\Pr(A_i | A_{i-1})$ is non-increasing in i, since it can easily be verified to be strictly less than 1 for $i = 3$. Let K_i be the set of strings such that $S_i \cup \{s\}$ is pairwise bounded if and only if $s \in K_i$, noting that $K_i = \emptyset$ if S_i is not pairwise bounded. We have $K_j \subseteq K_i$ for any $1 \leq i < j \leq n$. Since $\Pr(A_i | A_{i-1}) = \frac{|K_i|}{N_{\ell,2d}}$, the result holds. $\square$

Further, the probability that the set is not rejected is equal to $\left(\frac{N_{\ell,2d}}{|\Sigma|^\ell}\right)^{n-1}$. To empirically evaluate the efficiency of our rejection sampling method, we determined the portion of sets that will be rejected when generating a sample (of specified size) of pairwise bounded sets. We performed experiments with varying values of n, ℓ, and d, generated 10000 pairwise bounded sets in each experiment, and considered the average number of sets rejected before the pairwise bounded set was obtained. The default values for (n, ℓ, d) are $(20, 15, 4)$.

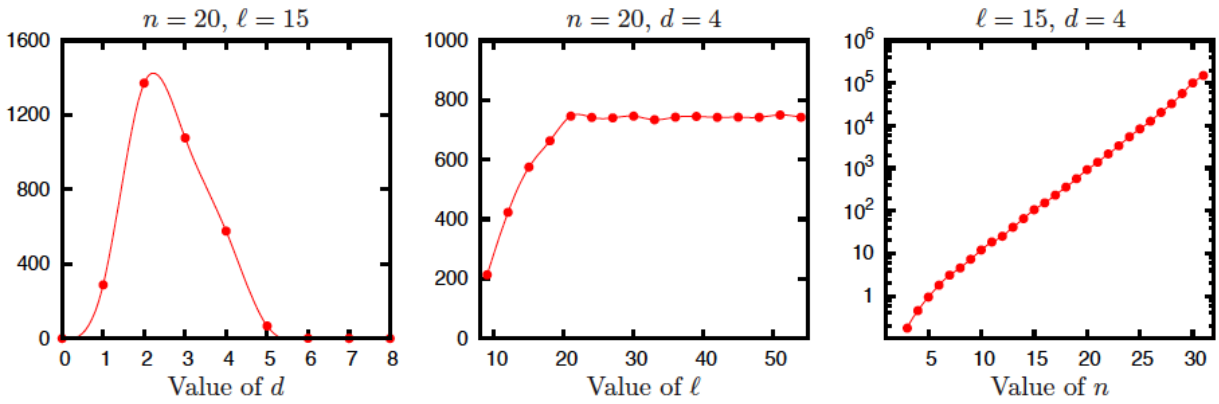

Figure 6.1: Data illustrating the mean number of sets rejected by our rejection sampling heuristic in order to generate a pairwise bounded set. Each plot shows the affect of varying one of the three parameters n, ℓ, and d. Data points are connected with cubic splines. Note the logarithmic scale used in the right plot.

The results of the empirical tests are shown in Figure 6.1. Each of the three plots shows how the average number of rejected sets changes when one of the three parameters is varied and the other two are fixed at their default values. The left plot shows what happens when d varies between 1 to 7. For values of d that are either greater than $\lfloor \ell/2 \rfloor$ or equal to 0, any set we generate is pairwise bounded and hence, we did not plot data for $d = 0$ or $d \geq 8$. The average number of rejected sets is largest when d is equal to 2 and decreases dramatically as d increases. This trend is expected since a large portion of non-pairwise bounded sets would be rejected when d is moderately large. The middle plot shows what happens when ℓ is varied between 9 and 55. The number of rejected sets increases steadily when ℓ varies within the range $[9, 20]$, then plateaus when ℓ is above 20. It can be easily shown analytically that increasing ℓ above $2dn$ will have no effect, however, we see empirically that the effect of ℓ is minimal for values of ℓ greater than 20. The right plot shows the effect of varying n between 3 and 31. Noting that a logarithmic scale is used, the average number of rejected sets exhibits growth that is clearly exponential in n.

6.1.1 A Separation of Weight Distributions

One of the key motivations for the development of methods to generate pairwise bounded sets from an appropriate distribution is that it can be used to determine whether there is a separation between the probability distribution of the weight of a random valid motif set and that of a random decoy set. We use the sampling method just described to generate 1000 random motif sets and 1000 random decoy sets for varying values of ℓ, d, and n. We ran the sampling algorithm described above to generate a pairwise bounded set then determined whether it is a motif set or a decoy set by using the dynamic programming algorithm described in Chapter 3. We continued generating pairwise bounded sets until we obtained 1000 decoy sets and 1000 motif sets. For each random motif and decoy set witnessed we calculated the weight of the set. Figure 6.2 depicts, for values considered for ℓ, d, and n, the distribution of the weight of the 1000 random motif sets and that of the 1000 random decoy sets. The data illustrate an adequate separation between the distributions.

As the value of n increases, the separation between the distributions becomes more prevalent since the probability distributions become more concentrated around their means and the means themselves diverge. Further, the dichotomy is again more evident when (ℓ, d) is increased from $(15, 4)$ to $(18, 6)$. When n is even moderately large we can use the weight to determine accurately whether the set is a motif set or a decoy set and as n increases this method of using the weight as an indicator will likely increase in accuracy. Similar conclusions can be made when ℓ and d increase. These results suggest that the simple heuristic of using the weight to determine whether a pairwise bounded set is a valid motif set or a decoy set will enable computationally challenging instances of the Closest String problem (*e.g.* when $n \geq 20$ or (ℓ, d) is equal to $(18, 6)$) to be solved efficiently with minimal probability of error.

These empirical trends illustrate the analytical results proved in Chapter 3 that demonstrate that the distribution of the weight of a random motif set is tightly concentrated around its mean. It is currently an open problem to prove an analogous result to Theorem 3.2.2 in Section 3.2.1 for an arbitrary decoy set. This is a considerably more challenging problem due to the lack of a combinatorial characterization of a decoy set.

6.2 An Overview of sMCL-WMR

sMCL-WMR considers a weighted graph representation of the input data (as MCL-WMR does), and then uses MCL) [110] to cluster the resulting graph. The construction of our graph $\mathcal{G}$ ensures that the motif instances represented by vertices in the graph are connected to each other and form a clique of size n, though the converse need not hold. Thus, the

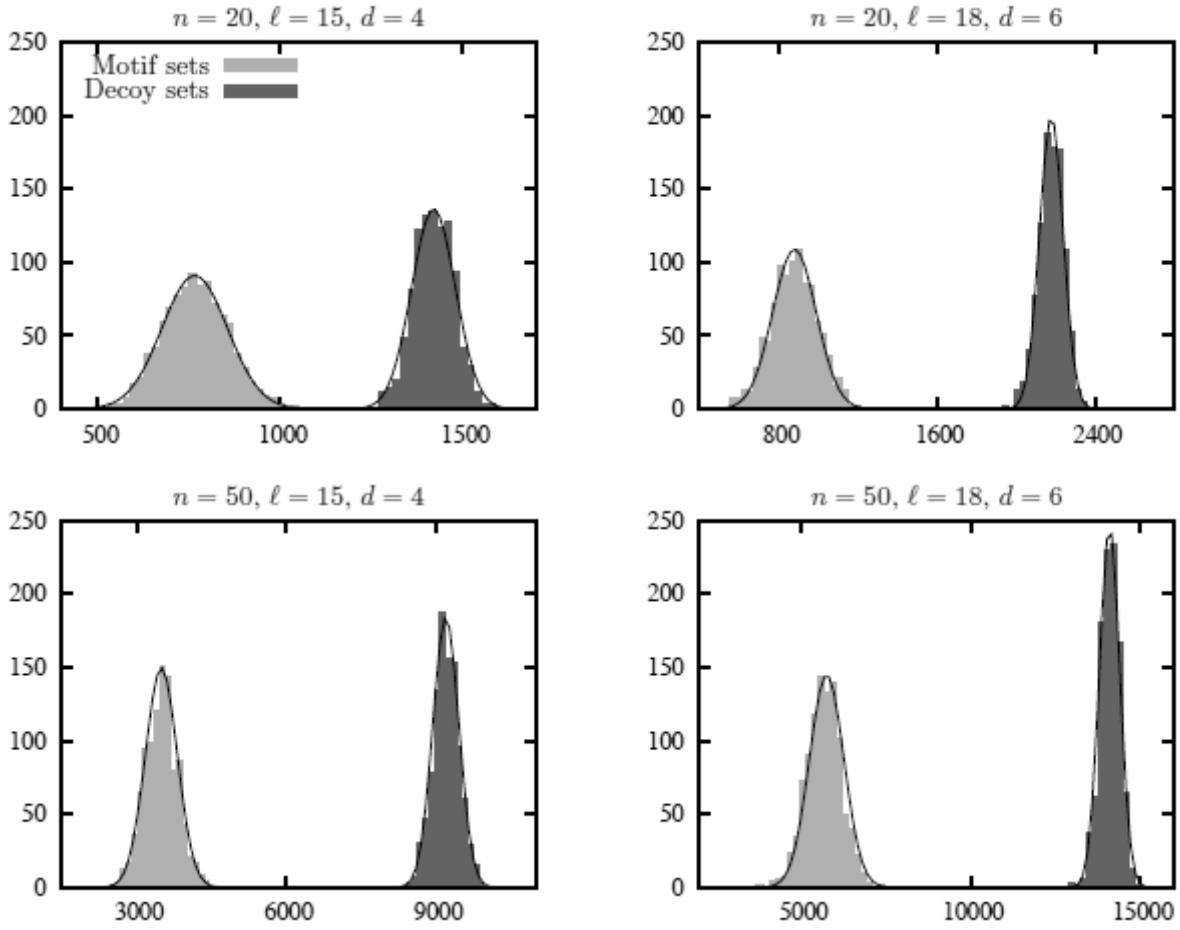

Figure 6.2: Data showing the distribution of the weight of a random motif set, and that of a random decoy set. Normal distributions fitted to the data are shown to indicate that the weight distributions are approximately normal.

problem of finding pairwise bounded sets in the data reduces to finding cliques of size n in the graph $\mathcal{G}$. Next, we filter out the clusters produced by MCL that do not meet the criteria of having at least n vertices or the minimum weight threshold. See Section 3.1 for the details of the construction of the graph and the graph clustering algorithm.

Figure 6.2 illustrates that both the weight of a random motif set and that of a random decoy set are approximately normally distributed, and shows a separation between these distributions. Using the rejection sampling method described earlier we calculate the mean and standard deviation of the weight of a random motif set and the weight of a random decoy set. We use $N(\mu, \sigma^2)$ to denote a normal distribution with mean μ and variance

σ^2. Let random variables W_m and W_d denote the weight of a random motif set and the weight of a random decoy set, respectively. Let μ_m and σ_m^2 respectively denote the mean and variance of the distribution of W_m and similarly, let μ_d and σ_d^2 respectively denote the mean and variance of W_d. Assuming that $W_m \sim N(\mu_m, \sigma_m^2)$ and $W_d \sim N(\mu_d, \sigma_d^2)$, we can determine the values α_m and α_d such that:

$$\Pr(W_m < \alpha_m) = .99 \text{ and } \Pr(W_d > \alpha_d) = .99 \ .$$

If $\alpha_m < \alpha_d$ then we can use the weight of a pairwise bounded set of strings to determine whether the set is a decoy or a motif as follows: calculate the weight w of the set and, if $w \leq \alpha_m$ or $w \geq \alpha_d$ then return that the set is a motif or a decoy, respectively; otherwise, use the dynamic-programming algorithm to classify the set. Hence, if $\alpha_m < \alpha_d$ then more than 99% of pairwise bounded sets will be classified correctly by considering the weight of the set. Typically the gap between α_m and α_d is large enough to guarantee that this rate is far higher than 99%. In theory it is possible that a set could be misclassified (*e.g.* if a motif set has weight greater than α_d) though in practice the probability of this happening is negligible and does not affect the performance of the algorithm.

(ℓ, d)	μ_m	μ_d	σ_m^2	σ_d^2	α_m	α_d
(15, 4)	794	1439	84	84	989	1243
(16, 5)	850	1651	86	102	1050	1413
(18, 6)	899	2204	89	140	1106	1878
(25, 8)	954	2670	111	175	1212	2262
(28, 9)	1024	3230	152	199	1378	2767
(30, 11)	1069	3882	169	245	1462	3312

Table 6.1: Data illustrating the mean and standard deviation of the weight of a random motif set and the weight of a random decoy set for various (ℓ, d)-motif problems. The number of strings is fixed at 20.

To increase the efficiency of sMCL-WMR, we include a pre-calculated table storing μ_m, μ_d, σ_m^2 and σ_d^2 for common values of ℓ, d, and n (for examples see Table 6.1 and 6.2). We varied n to be between 10 and 50, ℓ to be between 15 and 30, and d to be between $\lfloor \ell/5 \rfloor$ and $\lfloor \ell/2 \rfloor$. Values with weaker motifs or with small data sets (*i.e.* when $n \leq 10$) are not considered since it was shown that MCL-WMR performs efficiently for these instances.

n	μ_m	μ_d	σ_m^2	σ_d^2	α_m	α_d
15	432	980	52	60	552	840
20	794	1439	84	84	989	1243
25	1529	2250	129	110	1829	1994
30	1845	3263	196	169	2300	2869
35	2240	4523	246	213	2812	4027
40	3709	6110	389	275	4613	5460

Table 6.2: Data illustrating the mean and standard deviation of the weight of a random motif set and the weight of a random decoy set for various values of n. The values ℓ and d are fixed at 15 and 4, respectively.

6.3 An Overview of MCL-FSP

In many practical applications – including the analysis of the genetic data in this chapter – we are not only interested in identifying strings whose maximum distance from each the given strings is minimized but also in identifying strings whose minimum distance from each of the given strings is maximized. Given a set S_f of strings of length at least ℓ over an alphabet Σ and a non-negative parameter d_f, the objective of the Farthest String problem is to determine if there exists a string s over the alphabet Σ such that for any $s_i \in S_f$, $d(s, s_i) \geq d_f$. We refer to the subsequences occurring in the input sequences as *non-motifs*.

We describe a program, *MCL-FSP*, that given n length-m sequences over the alphabet Σ and parameters ℓ and d, finds substrings of length ℓ in the input data and a length-ℓ string s where the goal of the Farthest String problem is satisfied with respect to the parameters ℓ and d. MCL-FSP can be summarized by the following three steps: graph construction, graph clustering, and recovering the instances and their farthest string. The graph construction and clustering is similar to MCL-WMR and sMCL-WMR. However, the recovering of the substrings of interest is dramatically different. The graph constructed for MCL-FSP builds the same set of vertices but joins each pair of vertices by an edge if the Hamming distance between the strings corresponding to the pair of vertices is less than or equal to d. The weight on each edge is ℓ minus the Hamming distance between the corresponding strings of the endpoints of the edge. There exists no additional weighting on the edges.

The clustering of the graph will yield dense clusters in the graph. These dense subgraphs will likely contain a set of n substrings that are "close" – meaning the pairwise Hamming distances are small – and hence, are likely to have a string s, which satisfies the Farthest String problem together with the set of n substrings.

To recover sets on substrings that satisfy the Farthest String problem we first filter out clusters that do not have a substring from each of the input and clusters whose weight is less than $d \cdot \binom{n}{2}$. Clusters that pass this test may contain multiple cliques formed by choosing different subsets of n cluster vertices, or possibly no cliques at all. We identify all ways of forming a clique from the cluster vertices by using the n-partite nature of the graph to explore all possible cliques with a depth-first search. For each such clique, we use Algorithm 7 to determine whether the substrings in the clique correspond to a valid Farthest String solution.

Algorithm 7 FSP Recovery Algorithm

Input: A set of S n strings of length ℓ, parameters Δd and d, and a candidate string x.

Output: A string s^* with the minimum distance to any string in S at least d if it exists and "Not found" otherwise.

If $\Delta d < 0$ then return "Not found"

Choose $i \in \{1, \ldots, n\}$ such that $d(x, s_i) < d$. If no such i exists return x.

    $\mathcal{P} = \{p \,|\, x[p] = s_i[p]\}$;

    Choose any $\mathcal{P}'$ from $\mathcal{P}$ with $|\mathcal{P}'| = \ell - d + 1$.

    For each position $p \in \mathcal{P}'$

        Let x not be equal to s_i at position p

        s_{ret} = FSP Recovery Algorithm (S, $\Delta d - 1$, x)

        If $s_{ret} \neq$ "not found ", then return s_{ret}

Return "not found"

As mentioned previously, the Farthest String problem is NP-complete and therefore, unlikely to be solved in polynomial time. Algorithm 7 is based on the bounded search tree algorithm of Gramm *et al.* [57] for the Closest String problem, and has been previously studied by Cheng *et al.* [29]. Cheng *et al.* [29] proved Algorithm 7 has a worst-case running time of $O((|\Sigma|(\ell - d)^{\ell - d})$ and is guaranteed to solve Farthest String instances exactly.

6.4 Experimental Results on Synthetic Data

We tested sMCL-WMR and MCL-FSP on synthetic problem instances generated according to the embedded (ℓ, d)-motif model, and on real genetic data. The implementations of sMCL-WMR and MCL-FSP are coded in C++, and all experimental tests were performed on a Linux machine with a 64-bit 2600 MHz processor and 1 Gbyte of RAM running Ubuntu. The running time is given in CPU seconds.

We follow the experimental methods of Pevzner and Sze [88], and Buhler and Tompa [24] by considering the performance of sMCL-WMR and MCL-FSP in comparison to other

contemporary and well-known motif-recognition programs on synthetic data. We fix n to be equal to 20, m to be 600, and consider varied values of ℓ and d. To produce random motif-recognition instances, we generate a random center string of length ℓ, then generate n occurrences of the motif, each generated from the center string by randomly choosing d positions and for each of the d positions choosing a random replacement base from the four possible bases (A, C, G, T). We construct n background strings of length m and insert the generated motifs into a random position in the string. For each of the (ℓ, d) combinations, 100 randomly generated sets of input strings were generated.

We compared the performance of sMCL-WMR and MCL-FSP with that of the following motif-recognition programs: PROJECTION [24], MCL-WMR, PMSprune [35], and Voting [30]. All programs were run on the same Linux machine with the same data sets. These motif-recognition programs were chosen for their availability, performance, and widespread use; they are appropriate for comparison with sMCL-WMR because of the previously described capability in solving weak motif instances and because of their availability to be run on the described machine. The results of Voting, PMSprune, and PROJECTION are similar to the ones reported by Davila *et al.* [35], and to Chin and Leung [31], both of whose testing was completed on a machine with a slightly slower processor and the same core memory size.

(ℓ, d)	sMCL-WMR	MCL-FSP	MCL-WMR	PROJECTION	Voting	PMSprune
(10, 2)	122	1131	1020	560 (0.98)	30	42
(12, 3)	134	3019	2780	1921 (0.85)	124	130
(14, 4)	492	3325	3120	3058 (0.88)	562	556
(16, 5)	677	4502	4101	6132 (0.80)	2600	13121
(18, 6)	1521	8950	9202	-	8023	29012
(20, 7)	2845	-	-	-	24600	-
(25, 8)	4111	-	-	-	-	-

Table 6.3: Comparison of the performance of sMCL-WMR, MCL-FSP, and other motif-recognition programs on synthetic data for various values of ℓ and d. All programs except PROJECTION had a success rate of 1.0 and for this reason, the success rate was for PROJECTION is included in brackets in the table. In all experiments, $m = 1000$ and $n = 20$.

Tables 6.3 and 6.4 illustrate the comparison between the running time of sMCL-WMR and that of the other programs. Our aim was to test the selected programs on their capability to solve challenging motif instances (*i.e.* when d is significantly large with respect to ℓ). The symbol "-" implies that the program was not capable of solving the motif instance on the described machine in a reasonable amount of time, which we define to be at most 20 hours, or with reasonable accuracy, which we define to be at least 75%.

n	sMCL-WMR	MCL-FSP	MCL-WMR	PROJECTION	Voting	PMSprune
18	547	5559	5320	5230 (0.85)	3930	37020
20	679	8823	8912	-	5201	45030
24	1221	-	-	-	10211	-
28	2019	-	-	-	-	-
30	2431	-	-	-	-	-
40	5201	-	-	-	-	-

Table 6.4: Comparison of the performance of sMCL-WMR, MCL-FSP, and other motif-recognition programs on synthetic data for various values of n. All programs except PROJECTION had a success rate of 1.0 and for this reason, the success rate was for PROJECTION is included in brackets in the table.In all experiments, $\ell = 18$, $d = 6$ and $m = 1000$.

Two significant trends are witnessed in the data: sMCL-WMR is capable of solving very hard instances of motif recognition (*i.e.* when $\ell = 25$ and $d = 8$) and gives a dramatic improvement over the existing programs for instances where $\ell \geq 16$ (for instances where $\ell \leq 12$ sMCL-WMR had comparable or better performance to the other programs). MCL-FSP had a slightly higher running time than MCL-WMR and failed on the same instances as MCL-WMR.

6.5 Development of a Seed Coat-Specific Promoter for Canola

Canola (*Brassica napus L.*) was originally bred from rapeseed in Canada in the 1970s [41]. Presently, the canola industry generates more than $11 billion of yearly income to the Canadian economy. One of the major exports of the canola industry is *canola meal*, which is most widely used in animal feeds. However, one of the major problems with canola meal is the dark polyphenolic pigments that accumulate in the seed coat; the dark pigment interferes with the protein utilization, causing the quality of the meal to be lowered. Hence, altering the seed coat is an important biological challenge that has the possibility of reducing the indigestible fiber and enhancing the usability of canola meal. One first step in tackling this problem is to develop seed coat-specific promoters which are capable of regulating the genes involved in seed coat development and metabolism.

One approach to finding seed coat-specific promoters for canola is to isolate the promoters of proved seed coat-specific genes from other species, and then validate their expression in canola. Since genes are conserved among species, this is a reasonable approach. Preliminary results identified that several promoters express in the outer integument of seed

coat in canola, and a single promoter expresses in the inner integument. The subsequent challenge is to synthetically develop a promoter sequence that expresses both these two biological characteristics. This synthetic promoter can then be artificially inserted into the genomes of the canola seeds and be tested for their expression. Table 6.5 gives an overview of the relevant data concerning the promoter sequences analyzed.

Promoter	bp	Composition			
		A	C	G	T
AtLAC15	1528	33%	19%	16%	33%
Arabidopsis BAN	236	33%	18%	14%	34%
VPE	2067	34%	16%	17%	32%
GILT	2889	36%	13%	14%	37%
Arabidopsis TT12	1704	36%	16%	17%	31%
Arabidopsis TT2	3813	38%	14%	14%	34%
Barley Germin B gene	846	35%	15%	15%	35%
Tobacco Cryptic	2553	33%	18%	14%	34%
SCS1	7235	37%	12%	12%	39%
SCB1	5329	38%	12%	15%	35

Table 6.5: Description of promoters analyzed to develop a coat-specific promoter.

In order to use sMCL-WMR to find motifs in the genetic data we need to ensure that there exists a separation between the weight of the motif sets and decoy sets found in the data. We have previously shown such a separation exists for synthetic data. For a subset of the values of n, ℓ and d that are to be used in the analysis, we ran MCL-WMR and calculated the weight of the motif sets and decoy sets found. Figure 6.3 illustrates this data. For all values of n, ℓ, and d tested, there exists a clear separation between the distributions and we were able to use the precalculated tail values determined in Section 6.2.

The conserved motifs were identified through the use of sMCL-WMR and MCL-FSP. Table 6.6 gives a subset of the motifs detected by sMCL-WMR and the CPU time in seconds for their detection. In addition, it was of interest to identify furthest strings that were contained in one of the input sequences. Table 6.7 contains a sample of the furthest strings and which of the input sequences it occurs in; for the remaining sequences the specified string is a furthest string with respect to the parameters ℓ and d. In particular, detecting furthest strings that occurred in one of the following three promoters was of interest: the AtLAC15, Arabidopsis BAN and Arabidopsis TT2 promoters.

Each of these promoters has a specific biological role in the development of one of the different layers of seed coat. sMCL-WMR was used to determine the nucleotide patterns

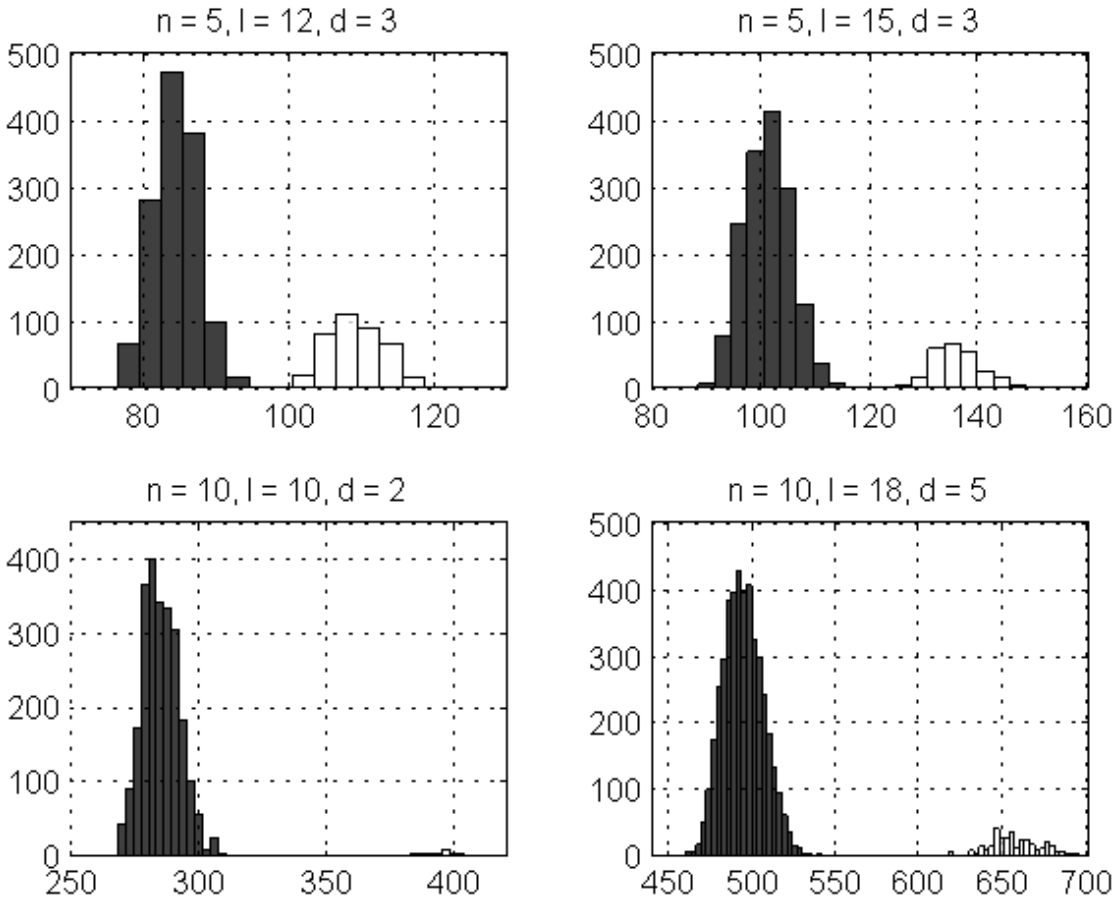

Figure 6.3: An Illustration of the distribution of the weight of a random motif set found in the promoter data (shown in white), and that of a random decoy set found in the promoter data (shown in black).

common to each of the promoters and hence, maybe responsible for having biological activity concerning the seed-coat. MCL-FSP was used to detect nucleotide patterns that were distinct to particular promoters that were responsible for the outer integument and inner integument of the seed coat – in hope of identifying biological sequence patterns responsible for each specific seed-coat integument. We identified more than 40 motifs and

non-motifs in the promoter data that may be responsible for the seed coat-specificity.

Based on these motifs and non-motifs, a promoter DNA sequence of approximately 700 bp was synthesized and introduced into canola. Presently laboratory tests are being used to obtain the complete expression results.

(ℓ, d)	n	Center String	CPU time
$(6, 1)$	10	ACActc	58
$(6, 2)$	10	tGgtCA	63
$(9, 1)$	9	taTCTttTT	98
$(9, 2)$	7	TTgtTAGgt	96
$(10, 2)$	8	TTTTtTattT	112
$(12, 3)$	10	tcTCTTtttCta	198
$(14, 4)$	7	AgtTctATTtttTT	253

Table 6.6: Subset of motifs detected using sMCL-WMR. The CPU time is in seconds.

(ℓ, d)	Occurrence	Furthest String	CPU time
$(10, 4)$	AtLAC15	ACCACTCCAG	1025
$(18, 8)$	AtLAC15	GATTTCCAAGCCTATCAC	1128
$(19, 9)$	AtLAC15	CCAAGAATCGATGAGCGGG	2591
$(15, 6)$	Arabidopsis BAN	GATCTACTGTTGTAC	1891
$(17, 7)$	Arabidopsis BAN	ATCACGTGCTTACCTTC	2139
$(15, 7)$	Arabidopsis TT2	CCGACGGGTTTGGCT	1811
$(18, 9)$	Arabidopsis TT2	CAGCGAAAAGGCCGACGG	2350

Table 6.7: Subset of motifs detected using MCL-FSP. The CPU time is in seconds.

6.6 Summary and Open Problems

We investigated the relationship between the weight of a decoy set and the weight of a motif set by means of random sampling. We discussed a rejection sampling strategy, and proposed a means to make this uniform sampling method more efficient. Using this algorithm that generates pairwise bounded sets uniformly at random, we studied the probability distributions of the respective weights of a random motif set and a random decoy set. We concluded that the weight of a pairwise bounded set can accurately predict whether the set is a valid motif set. We illustrated how to exploit this dichotomy to create a more efficient motif-recognition program.

Developing a more efficient method to generate pairwise bounded sets uniformly at random is an important algorithmic challenge that would give insight into the motif-recognition problem. Studying the possibility of the use of Markov chain Monte Carlo (MCMC) sampling algorithms to sample pairwise bounded sets warrants further investigation; such methods have led to efficient sampling algorithms for other combinatorial problems; see Randall [93] for a survey of MCMC methods and their applications. Further, an analytical explanation for the empirical results, showing that almost all decoy sets with degeneracy parameter d have center strings when the degeneracy allowed is increased to $d + 2$, would lead to a more efficient sampling method and would be interesting in its own right.

In addition, we developed and applied an efficient algorithm for the Furthest String problem. This problem is significantly less investigated than its partner problem, the Closest String problem, and as such, it warrants more in-depth study.

Chapter 7

Conclusion

This work focused on several fundamental problems of genomic sequence analysis: motif recognition, the Closest String problem, and other distinguishing string problems. We constructed predictive models for tasks in pattern recognition, and identified and applied many combinatorial and probabilistic insights to our problems of interest. Modelling biological problems as graphs and other abstract mathematical objects can lead to theoretical results concerning the computational complexity and thus, the ability to find an approximate solution efficiency. From a practical perspective, this area of research can lead to powerful new tools for identifying genes and possible mutations.

We described several original contributions – both theoretical and applied. Our main contributions are summarized below:

- We developed a new approach for motif recognition, and provided theoretical and experimental results that support our novel model and algorithm [16]. Our algorithm, MCL-WMR, builds a weighted graph model of the input data and uses a graph clustering algorithm to quickly determine important subgraphs that need to be searched further for valid motifs. Our experimental results show that MCL-WMR has competitive running time capabilities and accuracy.

- We gave a linear-time algorithm for solving Closest String instances with a small number of strings; which addressed an open problem of Gramm *et al.* [57]. We also considered the dual problem – instances with a large number of strings – and provided empirical results that demonstrate that these "large" instances can be solved efficiently. Our analytical explanation, as to why Closest String instances with a large number of strings are easily solved in practice, involved initiating the study of the smoothed complexity of the Closest String problem.

- We proposed a refined closest string model, the Closest String with Outliers (CSWO) problem, which asks for a center string s that is within Hamming distance d to at least $n - k$ of the n input strings, where k is a parameter describing the maximum number of outliers. A CSWO solution not only provides the center string as a representative for the set of strings but also reveals the outliers of the set. We gave fixed parameter algorithms for CSWO when d and k are parameters, for both bounded and unbounded alphabets. We also proved that when the alphabet is unbounded the problem is W[1]-hard with respect to $n - k$, ℓ, and d.

- Lastly, we applied the probabilistic heuristics and combinatorial insights for the Closest String problem to motif recognition. This program, referred to as *sMCL-WMR*, is used to uncover similarities in the promoter region of the genomic data of canola. We identified more than 40 motifs in the three promoters that might be responsible for certain biological activities concerning the seed coat, and synthetically developed a promoter that is conjectured to express all biological activities of interest. This synthetic promoter DNA sequence is currently being introduced into canola and tested for its expression.

Throughout this thesis we have suggested open problems that warrant further investigation. We conclude by giving further details on some of these suggested open problems, as well as proposing some future research directions.

Solving Small Motif Instances

In Chapter 4 we described a simple, linear-time algorithm for solving the Closest String problem. Since the development of this algorithm, these results have been extended by Amir *et al.* [3] to a variant of the optimization version of the Closest String problem that minimizes both the maximum Hamming distance but also the sum of (Hamming) distances from the strings to the center string. It is an open problem as to if there exists an efficient, polynomial-time algorithm for the Closest String problem restricted to four strings and an alphabet larger than the binary one, or for the Closest String problem restricted to a constant number of strings greater than four. In addition, extending the results of Amir *et al.* [3] to larger alphabets or sets of strings is currently open.

Smoothed Analysis

Numerous open problems still remain in the area of studying smoothed analysis and distinguishing string selection problems. Studying how robust the best (or good) instances rather than how fragile worst-case instances are warrants further consideration, as well as

studying the smoothed analysis of the PTAS of Li *et al.* [70]. The smoothed complexity of other distinguishing string selection problems remains open, these include the FARTHEST STRING problem and other variants of the CLOSEST STRING problem. Reconciling the complexity of these problems would further conclude about the ability to solve practical instances of variants to the CLOSEST STRING problem.

Practical Algorithmic Solutions for Solving CSWO

We gave several fixed parameter tractability algorithms for CSWO in Chapter 5. The practicality of these algorithms has not been investigated, and the worst-case analysis implies that they may be impractical for solving relatively large instances of this problem. Since there exist practical applications to this problem, it is imperative that algorithms for CSWO that are efficient and accurate in practice be developed.

Sampling and Counting Center Strings

In Chapter 6 we showed the applicability of sampling and counting center strings to motif recognition. There presently remain many open problems concerning this topic. Boucher and Omar gave results concerning the computational difficulty of sampling and counting center strings [19]. Developing efficient methods to sample and count center strings presently remains open. The development of a rapidly mixing *Markov chain Monte Carlo (MCMC)* algorithm could show the existence of an algorithm for uniform at random generation of pairwise bounded sets and should be further explored. MCMC methods have been successful in producing a sampling method for some combinatorial sampling problems. Extending Barvinok's algorithm [9], which counts the number of points in a polytope, to a weighted polytope would also give the desired result by combining it with the integer linear programming formulation of Boucher and Omar [19].

Bibliography

[1] L. Addario-Berry and B. Reed. Ballot theorems, old and new. *Bolyai Society Mathematical Studies*, 17, 2008. 74

[2] E.F. Adebiyi and M Kaufmann. Extracting common motifs under the levenshtein measure: theory and experimentation. In *Proceedings of Workshop on Algorithms in Biology (WABI)*, pages 140–156, 2002. 9

[3] A. Amir, G.M. Landau, J.C. Na, H. Park, K. Park, and J.S. Sim. Consensus optimizing both distance sum and radius. In *Proceedings of String Processing and Information Retrieval (SPIRE)*, pages 234–242, 2009. 87, 114

[4] A. Andoni, P. Indyk, and M. Patrascu. On the optimality of the dimensionality reduction method. In *Proceedings of Symposium on Foundations of Computer Science (FOCS)*, pages 449–456, 2006. 22, 36

[5] A. Andoni and R. Krauthgamer. The smoothed complexity of edit distance. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 357–369, 2008. 19, 62

[6] T.L. Bailey and C. Elkan. The value of prior knowledge in discovering motifs with MEME. In *Proceedings of International Conference on Intelligent Systems in Biology (ISMB)*, pages 21–29, 1995. 20, 23, 27, 34

[7] C. Banderier, R. Beier, and K. Mehlhorn. Smoothed analysis of three combinatorial problems. In *Proceedings of Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 198–207, 2003. 19, 62

[8] M.R. Barnes. *Bioinformatics for Geneticists: A Bioinformatics Primer for the Analysis of Genetic Data, 2nd Edition*. Wiley, 2007. 12

[9] A. Barvinok. Polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Mathematical Methods of Operations Research*, 19:769–779, 1994. 115

[10] A. Ben-Dor, G. Lancia, J. Perone, and R. Ravi. Banishing bias from consensus strings. In *Proceedings of Combinatorial Pattern Matching (CPM)*, pages 247–261, 1997. 12, 35, 89

[11] I. Ben-Gal, A. Shani, A. Gohr, A. Grau, J. Grau, S. Arviv, A. Shmilovici, S. Posch, and I. Grosse. Identification of transcription factor binding sites with variable-order Bayesian networks. *Bioinformatics*, 21(11):2657–2666, 2005. 7

[12] Y Bilu and N. Barkai. The design of transcription-factor binding sites is affected by combinatorial regulation. *Genome Biology*, 6(12):R103, 2005. 6

[13] A. Blum and J.D. Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In *Proceedings of ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 905–914, 2002. 19, 62

[14] B. Bollobas, S. Janson, and O. Riordan. The phase transition in inhomogeneous random graphs. *Random Structures and Algorithms*, 31(1):3–122, 2007.

[15] C. Boucher, D.G. Brown, and S. Durocher. On the structure of small motif recognition instances. In *Proceedings of Symposium on String Processing and Information Retrieval (SPIRE)*, pages 269–281, 2008. 2

[16] C. Boucher, P. Church, and D. Brown. A graph clustering approach to weak motif recognition. In *Proceedings of Workshop on Algorithms in Biology (WABI)*, pages 149–160, 2007. 2, 99, 113

[17] C. Boucher and J. King. Fast motif recognition via application of statistical thresholds. *BMC Bioinformatics*, 11(S1), 2010. 3

[18] C. Boucher and B. Ma. Closest String With Outliers. In *Submitted to the Asia Pacific Bioinformatics Conference (APBC)*, 2011. 3

[19] C. Boucher and M. Omar. On the hardness of counting and sampling center strings. In *Proceedings of Symposium on String Processing and Information Retrieval (SPIRE)*, pages 128–135, 2010. 115

[20] C. Boucher and K. Wilkie. Why large closest string instances are easy to solve in practice. In *Proceedings of Symposium on String Processing and Information Retrieval (SPIRE)*, pages 107–118, 2010. 2

[21] A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen. Predicting gene regulatory elements in silico on a genomic scale. *Genome Research*, 15:1202–1215, 1998. 34

[22] B. Brejová, D.G. Brown, I. Harrower, A. López-Ortiz, and T. Vinař. Sharper upper and lower bounds for an approximation scheme for CONSENSUS-PATTERN. In *Proceedings of Combinatorial Pattern Matching (CPM)*, pages 1–10, 2005. 10, 36, 88

[23] B. Brejová, D.G. Brown, I. Harrower, and T. Vinař. New bounds for motif finding in strong instances. In *Proceedings of Combinatorial Pattern Matching (CPM)*, pages 94–105, 2006. 10, 36, 88

[24] J. Buhler and M. Tompa. Finding motifs using random projections. *Journal of Computational Biology*, 2002. 2, 20, 28, 31, 34, 43, 48, 98, 99, 106, 107

[25] H.J. Bussemaker, H. Li, and E.D. Siggia. Building a dictionary for genomes: identification of presumptive regulatory sites by statistical analysis. *Proceedings of the National Academy of Sciences*, 97:10096–10100, 2000. 27

[26] M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64:165–171, 1997. 18

[27] J. Chen, X. Huang, I.A. Kanj, and G. Xia. Linear FPT reductions and computational lower bounds. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 212–221, 2004. 46

[28] Z.-Z. Chen, B. Ma, and L. Wang. A three-string approach to the closest string problem. In *Proceedings of the International Computing and Combinatorics Conference (COCOON) (to appear)*, 2010. 35, 93

[29] C.H. Cheng, C.C. Huang, S.Y. Hu, and K.-M. Chao. Efficient algorithms for some variants of the farthest string problem. In *Proceedings of Workshop on Combinatorial Math and Computation Theory*, pages 266–272, 2004. 36, 106

[30] Francis Y. L. Chin and C. M. Leung. Voting algorithms for discovering long motifs. In *Proceedings of Asia-Pacific Bioinformatics Conference (APBC)*, pages 261–271, 2005. 20, 32, 34, 107

[31] Francis Y. L. Chin and C. M. Leung. An efficient algorithm for string motif discovery. In *Proceedings of Asia-Pacific Bioinformatics Conference (APBC)*, pages 79–88, 2006. 20, 34, 107

[32] G. Cohen, I. Honkala, S. Litsyn, and P. Sole. Long packing and covering codes. *IEEE Transactions on Information Theory*, 43(5):1617–1619, 1997. 12

[33] F. Crick. Central dogma of molecular biology. *Nature*, 227:561–563, 1970. 4

[34] S. Crooke and B. Leblue. *Antisense research and applications*. CRC Press, 1993. 13

[35] J. Davila, S. Balla, and S. Rajasekaran. Fast and practical algorithms for planted (l, d) motif search. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):544–552, 2007. 20, 34, 107

[36] X. Deng, G. Li, Z. Li, B. Ma, and L. Wang. Genetic design of drugs without side-effects. *SIAM Journal on Computing*, 32(4):1073–1090, 2003. 13, 22, 23, 89

[37] P. D'haeseleer. What are DNA sequence motifs? *Nature Biotechnology*, 4(24):423–425, 2006. 23

[38] G. Dong and J. Pei. *Sequence Data Mining*. Springer, 2007. 7

[39] J. Dopazo, A. Rodríguez, J.C. Sáiz, and F. Sobrino. Design of primers for PCR amplification of highly variable genomes. *Computer Applications in the Biosciences*, 9:123–125, 1993. 13, 89

[40] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer, 1999. 16, 17

[41] R.K. Downey. Brassica oil seed breeding – achievements and opportunities. *Plant Breeding Abstracts*, 60(10):1165–1170, 1990. 108

[42] J.D. Dunagan, D.A. Spielman, and S-H. Teng. Smoothed analysis of the renegar's condition number for linear programming. In *Proceedings of SIAM Conference on Optimization (SIOPT)*, 2002. 62

[43] E. Eskin and P.A. Pevzner. Finding composite regulatory patterns in DNA strings. *Bioinformatics*, 18(1):354–363, 2002. 20

[44] P. A. Evans, A. Smith, and H. T. Wareham. On the complexity of finding common approximate substrings. *Theoretical Computer Science*, 306(1–3):407–430, 2003. 21

[45] M. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of motif search problems. *Combinatorica*, 26(2):141–167, 2006. 21, 89

[46] M.R. Fellows, J. Gramm, and R. Neidermeier. On the parameterized intractability of CLOSEST SUBSTRING and related problems. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 262–273, 2002. 89

[47] W. S. Feng, Z. Wang, and L. Wang. Identification of distinguishing motifs. In *Proceedings of Combinatorial Pattern Matching (CPM)*, pages 253–264, 2007.

[48] A. Floratos and I. Rigoutsos. On the time complexity of the TEIRESIAS algorithm. Technical Report Research Report RC 21161, IBM T.J. Watson Research Center, 1998. 9

[49] M. Frances and A. Litman. On covering problems of codes. *Theoretical Computer Science*, 30(2):113–119, 1997. 10, 12, 20

[50] D.J. Galas, M. Eggert, and M.S. Waterman. Rigorous pattern-recognition methods for DNA sequences. analysis of promoter sequences from Escherichia coli. *Journal of Molecular Biology*, 186(1):117–128, 1985. 32, 34

[51] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 16, 29, 46

[52] L. Gąsieniec, J. Jansson, and A. Lingas. Efficient approximation algorithms for the Hamming center problem. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 905–906, 1999. 35

[53] R.L. Graham and N.J.A. Sloane. On the covering radius of codes. *IEEE Transactions on Information Theory*, 31:385–401, 1985. 12

[54] J. Gramm, J. Guo, and R. Niedermeier. On exact and approximation algorithms for distinguishing substring selection. In *Proceedings of the International Symposium on Fundamentals of Computation Theory (FCT)*, pages 261–272, 2003. 23

[55] J. Gramm, J. Guo, and R. Niedermeier. Parameterized intractability of distinguishing substring selection. *Theoretical Computer Science*, 39(4):545–560, 2006. 23

[56] J. Gramm, R. Niedermeier, and P. Rossmanith. Exact solutions for CLOSEST STRING and related problems. In *Proceedings of International Symposium on Algorithms and Computation (ISAAC)*, pages 441–453, 2001. 52, 53, 54, 61, 77

[57] J. Gramm, R. Niedermeier, and P Rossmanith. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37(1):25–42, 2003. 2, 10, 35, 36, 53, 54, 55, 56, 57, 62, 70, 71, 77, 87, 89, 91, 92, 93, 106, 113

[58] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. 12

[59] G.H. Hertz, G.W. 3rd Hartzell, and G.D. Stormo. Identification of consensus patterns in unaligned DNA sequences known to be functionally related. *Computer Applications in the Biosciences*, 6(2):81–92, 1990. 27, 34

[60] G.H. Hertz and G.D. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15:563–577, 1999. 23, 27, 34

[61] E.M. Hillis, C. Moritz, and B.K. Mable. *Molecular Systematics*. Sinauer Associates Inc., 1996. 13

[62] F. Jacob and J. Monod. Genetic regulatory mechanisms in the synthesis of proteins. *Journal of Molecular Biology*, 3(3):318–356, 1961. 4

[63] D.S. Johnson. Approximation algorithms for combinatorial problems. *journal of computer and system sciences*, 9:256–278, 1974. 18

[64] L. Kaderali and A. Schliep. Selecting signature oligonucleotides to identify organisms using DNA arrays. *Bioinformatics*, 18(10):1340–1349, 2002. 12

[65] A. Krause, M. Kräutner, and H. Meier. Accurate method for fast design of diagnostic oligonucleotide probe sets for DNA microarrays. In *Proceedings of the International Symposium on Parallel and Distributed Processing*, pages 154–163, 2003. 12

[66] J.K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 633–642, 1999. 11, 35

[67] J.K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, pages 41–55, 2003. 9, 10, 12, 13, 21, 35, 36, 89

[68] C.E. Lawerence, 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*, 262:208–214, 1993. 23, 25, 34

[69] W.H. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983. 35, 93

[70] M. Li, B. Ma, and L. Wang. Finding similar regions in many strings. *Journal of Computer and System Sciences*, 65(1):73–96, 2002. 10, 21, 35, 88, 89, 115

[71] S. Liang, M.P. Samanta, and B.A. Biegel. cWINNOWER algorithm for finding fuzzy DNA motifs. *Journal of Bioinformatics and Computational Biology*, 2(1):47–60, 2004. 30, 31

[72] X. Liu, D.L. Brutlag, and J.S. Liu. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. In *Proceedings of Asia Pacific Bioinformatics Conference (APBC)*, pages 127–138, 2001. 26, 27

[73] X.S. Liu, D.L. Brutlag, and J.S. Liu. An algorithm for finding protein-DNA binding sites with applications to chromatin-immunoprecipitation microarray experiments. *Nature Biotechnology*, 20(8):835–839, 2002. 23, 27

[74] K. Lucas, M. Busch, S. Össinger, and J.A. Thompson. An improved microcomputer program for finding gene- and gene family-specific oligonucleotides suitable as primers for polymerase chain reactions or as probes. *Computer Applications in the Biosciences*, 7:525–529, 1991. 13, 89

[75] B. Ma. A polynomial time approximation scheme for the closest substring problem. In *Proceedings of Combinatorial Pattern Matching (CPM)*, pages 99–107, 2000. 9, 21, 22, 89

[76] B. Ma. Why greedy works for shortest common superstring problem. In *Proceedings of Combinatorial Pattern Matching (CPM)*, pages 244–254, 2008. 19, 62

[77] B. Ma and X. Sun. More efficient algorithms for closest string and substring problems. In *Proceedings of Research on Computational Biology (RECOMB)*, pages 396–409, 2008. 10, 21, 22, 35, 36, 89, 92, 93

[78] B. Ma, J. Tromp, and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002. 10, 19, 62

[79] A. Macario and E. Macario. *Gene Probes for Bacteria*. Academic Press, 1990. 12

[80] B. Manthey and R. Reischuk. Smoothed analysis of binary search trees. *Theoretical Computer Science*, 378(3):292–315, 2007. 19, 62, 88

[81] H.M. Martinez. An efficient method for finding repeats in molecular sequences. *Nucleic Acids Research*, 11(13):4629–4634, 1983. 9, 34

[82] D Marx. The closest substring problem with small distances. In *Proceedings of Symposium on Foundations of Computer Science (FOCS)*, pages 63–72, 2005. 21

[83] R. Motwani and R. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. 44, 45

[84] R. Niedermeier. *Invitation to fixed-parameter algorithms*. PhD thesis, Universität Tübingent, 2002. 46

[85] C.H. Papadimitriou. On selecting a satisfying truth assignment. In *Proceedings of Foundation of Computer Science (FOCS)*, pages 163–169, 1991. 63

[86] G. Pavesi, G. Mauri, and G. Pesole. An algorithm for finding signals of unknown length in DNA sequences. *Bioinformatics*, 17:S207–S214, 2001. 89

[87] 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(W):199–203, 2004. 20

[88] P. Pevzner and S. Sze. Combinatorial approaches to finding subtle signals in DNA strings. In *Proceedings of International Conference on Intelligent Systems in Biology (ISMB)*, pages 269–278, 2000. 2, 9, 20, 28, 29, 30, 31, 33, 34, 36, 37, 43, 46, 89, 99, 106

[89] N. Pisanti, A.M. Carvalho, L. Marsan, and M-F. Sagot. RISOTTO: Fast extraction of motifs with mismatches. In *Proceedings of Latin American Theoretical Informatics Symposium (LATIN)*, pages 757–768, 2006. 20

[90] A. Price, S. Ramabhadran, and P.A. Pevzner. Finding subtle motifs by branching from sample strings. *Bioinformatics*, 19(S2):149–155, 2003. 33, 34

[91] V. Proutski and E.C. Holme. Primer master: A new program for the design and analyiss of PCR primers. *Computer Applications in the Biosciences*, 12:253–255, 1996. 13, 89

[92] S. Rajasekaran, S. Balla, and C. H. Huang. Exact algorithms for planted motif problems. *Journal of Computational Biology*, 12(8):1117–1128, 2005. 9, 20, 29, 34

[93] D. Randall. Rapidly mixing Markov chains with applications in computer science and physics. *IEEE Computing in Science and Engineering*, 8(2):30–41, 2006. 112

[94] E. Rocke and M. Tompa. An algorithm for finding novel gapped motifs in DNA sequences. In *Proceedings of Research on Computational Biology (RECOMB)*, pages 228–233, 1998. 32

[95] F.P. Roth, J.D. Hughes, P.W. Estep, and G.M. Church. Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. *Nature Biotechnology*, 16(10):939–945, 1998. 26

[96] M.F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *Proceedings of Latin American Theoretical Informatics Symposium (LATIN)*, pages 374–390, 1998. 9

[97] F. Sanger, S. Nicklen, and A.R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12), 1977. 1

[98] R.V. Satya and A. Mukherjee. A new algorithm for finding monad patterns in DNA sequences. In *Proceedings of Symposium on String Processing and Information Retrieval (SPIRE)*, pages 273–285, 2004. 30, 31

[99] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of Foundation of Computer Science (FOCS)*, pages 410–414, 1999. 63

[100] S. Sinha and M. Tompa. A statistical method for finding transcription factor binding sites. In *Proceedings of International Conference on Intelligent Systems for Molecular Biology (ISMB)*, volume 8, pages 344–354, 2000. 34

[101] S. Sinha and M. Tompa. YMF: a program for discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucleic Acids Research*, 31(13):3586–3588, 2003. 23, 27

[102] A. Smith. *Common Approximate Substrings*. PhD thesis, University of New Brunswick, 2004. 9, 12, 21, 22, 52, 90, 97

[103] D.A. Spielman and S-H Teng. Smoothed analysis of algorithms: why the simplex algorithm ususally takes polynomial time. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 296–3051, 2001. 2, 19

[104] R. Staden. Methods for discovering novel motifs in nucleic acid sequences. *Computer Applications in the Biosciences*, 5(4):262–271, 1989. 34

[105] N. Stojanovic, P. Berman, D. Gumucio, R. Hardison, and W. Miller. A linear-time algorithm for the 1-mismatch problem. In *Proceedings of Workshop on Algorithms and Data Structures (WADS)*, pages 126–135, 1997. 35

[106] M.P. Styczynski and K.L. Jensen. An extension and novel solution to the $(1, d)$-motif challenge problem. *Genome Informatics*, 15(2):63–71, 2004. 47

[107] S. Sze, S. Lu, and J. Chen. Integrating sample-driven and patter-driven approaches in motif finding. In *Proceedings of Workshop on Algorithms in Biology (WABI)*, pages 438–449, 2004. 20, 30, 31, 52, 53

[108] M. Tompa. An exact method for finding short motifs in sequences, with application to ribosome binding site problem. In *Proceedings of International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 262–271, 1999. 34

[109] M. Tompa, N. Li, T. L. Bailey, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, W. J. Kent, et al. Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, 23(1):137–144, 2005. 10, 86, 87, 89

[110] S. van Dongen. *Graph clustering by flow simulation*. PhD thesis, University of Utrecht, May 2000. 38, 39, 47, 102

[111] J. van Helden, B. Andre, and J. Collado-Vides. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *journal of molecular biology*, 281:827–842, 1998. 27

[112] J. van Helden, B. Andre, and J. Collado-Vides. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Journal of Molecular Biology*, 281(5):827–842, 1998. 34

[113] V. Vazirani. *Approximation Algorithms*. Springer, 2003. 18

[114] J. Vilo, A. Brazma, I. Jonassen, A. Robinson, and E. Ukkonen. Mining for putative regulatory elements in the yeast genome using gene expression data. In *Proceedings of the International Conference on Intelligent Systems in Molecular Biology (ISMB)*, pages 384–394, 2000. 27

[115] L. Wang and B. Zhu. Efficient algorithms for the closest string and distinguishing string selection problems. In *Proceedings of the International Workshop on Frontiers in Algorithmics (FAW)*, pages 261–270, 2009. 35, 36, 93

[116] M. Waterman, R. Arratia, and E. Galas. Pattern recognition in several sequences:consensus and alignment. *Bulletin of Mathimatical Biology*, 46:515–527, 1984. 20, 32

[117] E. Wingender, P. Dietze, H. Karas, and R. Knüppel. TRANSFAC: a database on transcription factors and their DNA binding sites. *Nucleic Acids Research*, 24(1):238–241, 1996. 86

[118] X. Yang and J. Rajapakse. Graphical approach to weak motif recognition. *Genome Informatics*, 15(2):52–62, 2004. 30, 31, 46

[119] R. Zhao and N. Zhang. A more efficient closest string problem. In *Proceedings of Bioinformatics and Computational Biology (BICoB)*, pages 210–215, 2010. 35, 93