

Common Approximate Substrings

by

Andrew David Smith

B.A., B.C.S., University of New Brunswick, 2000

A Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of

Doctor of Philosophy

in the Graduate Academic Unit of Computer Science

Supervisor: Patricia A. Evans, Ph.D., Computer Science

Examining Board: Joseph D. Horton, Ph.D., Computer Science, Chair
Bradford G. Nickerson, Ph.D., Computer Science
Maryhelen Stevenson, Ph.D.,
Electrical and Computer Engineering

External Examiner: Ming Li, Ph.D., School of Computer Science,
University of Waterloo

This thesis is accepted.

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

January, 2004

(C) Andrew David Smith, 2004

ABSTRACT

Discovering patterns in strings is a central task in analyzing molecular sequences. One pattern discovery problem is to find a pattern that occurs as a substring in each member of a given set of strings. Additionally, occurrences of this pattern are allowed to have up to some specified number of errors, so the occurrences may not exactly match the pattern. Allowing such “approximate occurrences” significantly complicates methods for discovering the pattern, rendering it NP-hard.

The pattern discovery problem is abstracted as a decision problem under the name Common Approximate Substring. A systematic parameterized complexity analysis is conducted, producing a nearly complete parameterized complexity map with respect to the number of input sequences, their maximum length, the length of the pattern, the maximum number of mismatches between the pattern and its occurrences, and the size of the sequence alphabet. The analysis has also revealed several new results, including the first FPT variant not parameterized with alphabet size.

The Closest Substring problem is an optimization problem with the goal of minimizing the maximum number of mismatches between the pattern and any occurrence. Previous studies did not resolve the effect of alphabet size on approximability. The problem admits a polynomial time approximation scheme even for the case of an unrestricted alphabet, and an improvement is given for the case of a binary alphabet. Objective functions are derived from three other aspects of the problem, and new approximability results are described for each associated optimization problem. The results include both upper and lower bounds, and in one case these bounds are shown to be tight.

In practice one might only know some set of necessary conditions for important patterns, and desire the identity of each pattern matching those conditions. This goal is captured by the problem of enumerating all patterns fitting a specified length and number of allowed errors. New upper bounds are proved for the complexity of the enumeration problem, and parallelizations are described that have both practical and theoretical value. The upper bound on the complexity is achieved through the use of a new data structure capable of concisely encoding sets of patterns, while providing efficient membership queries and efficient set operations.

Table of Contents

Abstract	ii
List of Tables	vi
List of Figures	vii
List of Algorithms	viii
List of Symbols	x
1 Introduction	1
1.1 Motivation and Objectives	1
1.2 Contributions	1
1.3 Thesis Overview	3
2 Background	4
2.1 Computational Complexity Theory	4
2.1.1 Parameterized complexity	7
2.1.2 Approximation theory	11
2.2 Molecular Sequence Patterns	13
2.2.1 Review of molecular biology	13
2.2.2 The nature of sequence patterns	15
2.2.3 Operational definition of “pattern”	16
2.2.4 Importance of sequence pattern discovery	18
2.3 General Properties of the Central Problem	21
3 Parameterized Complexity Analysis	26
3.1 Fixed Parameter Tractable Variants	26

3.2	Parameterized Hardness	30
3.2.1	W[1]-hardness of $CAS(m, l)$	31
3.2.2	W[2]-hardness of $CAS(l)$	34
3.2.3	W[2]-hardness of $CAS(m, \Sigma)$	37
3.2.4	Other hardness results	41
3.3	Membership in the W-Hierarchy	42
3.4	Summary and Open Problems	46
4	Optimizing Aspects of Patterns	49
4.1	Closest Substring and Alphabet Size	50
4.1.1	Additional terminology	51
4.1.2	Assuming constant sized alphabet	53
4.1.3	Eliminating alphabet size dependence	54
4.1.4	Stronger results for binary strings.	62
4.1.5	Larger constant sized alphabets	63
4.2	Approximating Alternate Objectives	66
4.2.1	The similarity aspect	67
4.2.2	The quorum aspect	70
4.2.3	The length aspect	74
4.3	Connection with Parameterized Complexity	77
4.3.1	Parameterized approximation	77
4.3.2	Hardness by parameterized reductions	79
4.4	Summary and Open Problems	79
5	Toward Optimal Enumeration	81
5.1	Improved Time and Space Complexity	82
5.1.1	The Census algorithm	83
5.1.2	Complexity analysis	85
5.1.3	Parallelizations	88
5.2	Further Improvements	90
5.2.1	Simulating a boolean circuit	90
5.2.2	Neighborhood trees for set operations	92
5.2.3	A PRAM algorithm	96
5.3	Additional Applications	98
5.3.1	The Substring Parsimony problem	98
5.3.2	The Distinguishing Substring problem	100
5.4	Summary and Open Problems	101

6 Conclusion	103
Bibliography	105
A Circuit Diagrams	114

List of Tables

- 3.1 Parameterized complexity map 47
- 4.1 Approximability of CLOSEST SUBSTRING problems 80
- 5.1 Time and space complexity of finding exact solutions 101

List of Figures

3.1	Example graph 1	32
3.2	CAS(m, l) representation for example graph 1	32
3.3	Example graph 2	35
3.4	CAS(l) representation for example graph 2	36
3.5	Example instance of SET COVER.	38
3.6	CAS($m, \Sigma $) representation for SET COVER instance.	39
3.7	Substrings corresponding to covering set.	40
A.1	Configuration of the instance testing circuit.	115
A.2	Configuration of the center testing circuit.	116
A.3	High level configuration of the single instance + modifications testing circuit.	117
A.4	Mismatch counting part of the single occurrence + modifications testing circuit.	118
A.5	Inputs to the single occurrence + modifications testing circuit.	119

List of Algorithms

1	VALID	24
2	SCORE	25
3	DEVELOPCENTER	29
4	LARGEALPHABET	61
5	GREEDYANDLAZY	64
6	GREEDYMAXIMUMCOVERAGE	73
7	EXTENDOCCURRENCE	75
8	BOUNDENUMERATION	78
9	CENSUS	84
10	CIRCUITSIMULATION	91
11	BUILDNEIGHBORHOOD	94
12	PARALLELCIRCUITSIMULATION	97

List of Symbols

$N_{a,b}$	(a, b) -neighborhood, page 23
Σ	alphabet, page 22
Σ_R	alphabet restricted to positions of R , page 53
\circ	concatenation, page 93
$\exp(x)$	base of the natural logarithm raised to power x , page 64
\leq_{GP}	the relation of gap preserving reducibility, page 13
opt	optimal value of an objective function, page 11
\prod	arithmetic product, logical conjunction, page 45
\sum	arithmetic sum, logical disjunction, page 45
d_H	Hamming distance function, page 22
CAS	Common Approximate Substring, page 26

Complexity Classes

log-APX	class of constant factor approximable problems, page 12
poly-APX	class of problems approximable within a polynomial factor, page 12
FPT	Fixed Parameter Tractable, Class of fixed parameter tractable problems, page 8
FPTAS	Fully Polynomial Time Approximation Scheme, page 12

NP	Nondeterministic Polynomial Time, class of problems solvable in non-deterministic polynomial time, page 6
P	Polynomial Time, class of problems solvable in deterministic polynomial time, page 6
PTAS	Polynomial Time Approximation Scheme, page 12
$W[x]$	class of parameterized problems expressible as parameterized circuits of weft x , page 10
XP	class of parameterized problems solvable in time $O(n^{f(k)})$, where n is the input size and k is the parameter, page 10

Problem Specific

d	maximum Hamming distance between center and occurrence, page 22
d_{opt}	optimal value of the objective function for a Closest Substring instance, page 50
l	length of a center or motif that is a solution to a pattern discovery problem, page 22
m	size of sequence set in an instance of a pattern discovery problem, page 22
N	size of the (l, d) -neighborhood of a length l string, page 23
n	length of sequences in an instance of a pattern discovery problem, page 22
\mathcal{C}	symbol designating a center for an instance of a pattern discovery problem, page 22
\mathcal{F}	family of sequences for an instance of a pattern discovery problem, page 22

Chapter 1

Introduction

1.1 Motivation and Objectives

Until recently, research in molecular biology has been hindered by the limited availability of data. In the post-genomic age biologists have access to raw data with vast potential for discovery. From such large amounts of data come new challenges of extracting the resident “biological meaning”. This undiscovered knowledge exists presently in sequence databases and unquestionably holds key information needed to solve important problems. There is also no question that the present technology for extracting information from raw sequence data is insufficient for the task. The recent paradigm shift, caused by the increased availability of data, has forced biology to rely on discrete mathematics and computer science in the way it has traditionally relied on chemistry, physics, and traditional statistical methods.

This thesis examines some theoretical and practical issues surrounding techniques for extracting information from bio-molecular sequences. The specific problem analyzed is that of discovering contiguous patterns that occur approximately in each member of a set of strings. By “occurring approximately”, we mean that patterns must match a substring of each string with at most some specified number of mismatches. The problem is important as an abstraction for many tasks faced by biologists attempting to discover information in molecular sequences. Computational solutions to problems of this sort are presently inadequate. The aim of this thesis is to provide an improved understanding of the pattern discovery problem by applying tools from computational complexity theory.

1.2 Contributions

The central problem examined in this thesis is analyzed from three perspectives, each with a slight variation on the problem definition. The focus is theoretical,

but the results provide important knowledge concerning practical solutions. The following list describes the contributions of this thesis according to the three perspectives from which the problem is analyzed.

- A systematic parameterized complexity analysis was conducted, investigating the properties of five different problem aspects: the number of sequences in which to discover a pattern, the length of those sequences, the length of the pattern, the size of the sequence alphabet, and the maximum distance between pattern and pattern occurrence.

This analysis has demonstrated the first fixed parameter tractable variant of the problem that does not require both a fixed sized alphabet and a fixed pattern length. An algorithm is described that indicates the sufficiency of fixing the length of the input sequences regardless of the number of sequences or the size of the sequence alphabet.

Negative complexity results include two original $W[2]$ -hardness proofs. These establish the extreme intractability of two important variants. In addition to negative results for the W -hierarchy, several parameterized circuit families have been designed, demonstrating membership in various classes, and providing alternate characterizations of problem solutions.

- Approximability of the pattern discovery problem has been studied under the name Closest Substring, and strong approximability results have been obtained for the problem when restricted to a constant sized alphabet. In this thesis the first polynomial time approximation scheme is given for the problem with no restrictions on the size of the input alphabet. In addition, significant strengthening of known results are given for the special case of a binary input alphabet.

Alternate objectives are analyzed, including the *similarity* between pattern and pattern occurrence, the number of input sequences containing a common pattern, and the length of a common pattern. Several results are given for these problems showing that none are as easily approximated as the original Closest Substring problem.

- The pattern discovery problem is also investigated as to the complexity of producing all patterns that fit the requirements of the problem instance. This treatment is closer to the actual situation faced in the biological research setting. An algorithm is described that improves on one of the best

existing algorithms and simultaneously gives new upper bounds on the complexity of the corresponding decision problem. Various parallelizations, both practical and theoretical, are described.

1.3 Thesis Overview

This thesis is organized around the systematic investigation of the central pattern discovery problem. Chapter 2 presents background information concerning complexity theory and relevant concepts concerning biological sequences.

In Chapter 3 the problem is abstracted as a decision problem under the name *COMMON APPROXIMATE SUBSTRING*. The analysis and results are divided into tractability (Section 3.1), hardness (Section 3.2), and complexity class membership (Section 3.3).

Under the name *CLOSEST SUBSTRING*, in Chapter 4 the problem is investigated with the tools of approximation theory. In Section 4.1, the influence of alphabet size is considered. In Section 4.2 focus turns to alternate objective functions, and their complexity of approximation.

In Chapter 5 a more practical perspective is taken. The problem is abstracted as the *MOTIF ENUMERATION* problem, for which a solution is the entire set of valid patterns. In Section 5.1, a sequential algorithm is given; the algorithm improves on the complexity of previous algorithms by a factor of the number of input sequences. In Section 5.2, further improvements are given, including a new data structure for encoding sets of motifs, and supporting efficient set operations and membership queries. In addition to improvements in the serial time complexity, the algorithm is modified to run on a *PRAM* in logarithmic time with a polynomial number of processors.

The discussion and results in this thesis are of a highly technical nature. The descriptions and analysis of algorithms and complexity results are intended to be accessible to computer scientists, mathematicians and computational biologists. The intent is that results in this thesis will guide the development of algorithms for solving related pattern finding problems. The complexity results indicate possible techniques to attempt (and some to avoid) while searching for better algorithms. The algorithms and heuristics developed here can provide the basis for more powerful and general programs.

Chapter 2

Background

This chapter is divided into sections. Section 2.1 provides a review of complexity theory, including detailed background material on parameterized complexity and the theory of approximation algorithms. Section 2.2 introduces the concepts of molecular biology necessary to appreciate the motivation for this research. Following a discussion of previous definitions, an operational definition for the patterns is developed, followed by a discussion of certain applications of the problem. In Section 2.3, a formal definition is given for the pattern discovery problem as a decision problem called COMMON APPROXIMATE SUBSTRING, and certain fundamental results are presented concerning instances of this problem.

2.1 Computational Complexity Theory

This section presents definitions to facilitate the communication of ideas in later chapters. This thesis centers around a specific computational problem. The following is an informal description from Garey and Johnson’s “Computers and Intractability: A Guide to the Theory of NP-Completeness” [36].

For our purposes, a *problem* will be a general question to be answered, usually possessing several *parameters*, or free variables, whose values are left unspecified. A problem is described by giving: (1) a general description of all its parameters, and (2) a statement of what properties the answer, or *solution*, is required to satisfy. An *instance* of a problem is obtained by specifying particular values for all the problem’s parameters.

As a mathematical concept, a problem is a type of relation.

Definition 2.1.1 (problem) A problem Π is defined as a relation $\Pi \subseteq I \times S$, where I is the set of *instances* and S is the set of *solutions*.

This thesis examines decision problems, optimization problems, and search problems.

Definition 2.1.2 (decision problem) A decision problem Π is any problem with a Boolean solution set, *i.e.* $S = \{\text{true}, \text{false}\}$. For simplicity, the set Π is often used to refer to those instances $I' \subset I$ such that $(I', \text{true}) \in \Pi$.

Search problems bear a closer correspondence to the real world problems that motivate computational problems.

Definition 2.1.3 (search problem) A search problem Π is a problem for which a solution is some object bearing certain specified properties, or the assertion that no such object exists. In other words, Π has solution set $S = S' \cup \{\text{false}\}$, where S' is a finite set of finite objects.

Optimization problems are supplemented with an objective function that must be maximized or minimized, depending on the problem¹.

Definition 2.1.4 (optimization problem) An optimization problem Π is a problem with a set of feasible solutions and the objective of maximizing or minimizing some objective function that maps feasible solutions to some set of rational numbers.

An algorithm is a finite procedure for solving a problem. Algorithms are described throughout this thesis using pseudocode with control structures commonly found in modern programming languages, and mathematical notation to express data types and operations. The quality of an algorithm is usually based on resource requirements, or complexity, in terms of both time and space; an algorithm is assessed in terms of its time and space complexity. The theoretical discussions in this thesis deal mainly with worst case complexity, but often the requirements of an algorithm on average or expected cases are important.

Problems are classified according to their computational complexity, or the complexity of known algorithms for solving them. A complexity class is any set of problems for which methods of obtaining a solution share some specific property (usually related to resource requirements). The model of computation is not an issue in this thesis, but it is assumed to be the Turing Machine model during

¹An interesting difference between maximization problems and minimization problems is how often their descriptions in second order logic differ [52]. It has been conjectured that these differences in descriptive complexity account for the differences in approximability of complementary problems.

theoretical discussions and the Random Access Machine model when dealing with subexponential time computation.

As first recognized by Edmonds [27], the most important complexity classes are the classes of problems solvable in time that is bounded by a polynomial function of the size of the input. A distinction is made between those problems solvable in polynomial time on deterministic and non-deterministic machines. A deterministic machine must obtain a solution, whereas a non-deterministic machine can *guess* a solution and is always correct provided a solution exists. The work done by the non-deterministic machine is simply what is required to verify that the guessed solution is, in fact, correct. The class of problems solvable in polynomial time on a deterministic machine is denoted P and the class of problems solvable in polynomial time on a non-deterministic machine is denoted NP. It is important to keep in mind that non-deterministic machines exist only as mathematical tools. The relation between these classes is known to be

$$P \subseteq NP.$$

For many years it has been generally believed that the inclusion is proper but this is yet to be proved [22].

Originally, the evidence for $P \neq NP$ was that a large number of problems known to be in NP had no known polynomial time algorithms. Cook [21] proved that if the problem of deciding the satisfiability of Boolean formulas is in P, then so is every other problem in NP. Shortly after, Karp [50] used this result and showed that many other well known problems in NP share that same property. These problems are called NP-complete, meaning that they are in NP, and a polynomial time algorithm for any one of them can be used to construct a polynomial time algorithm for all problems in NP. Cook's result essentially acted as a bootstrap for the theory; Karp used Cook's theorem in conjunction with *polynomial time reductions* to make the theory useful. The definition given below for "reduction" technically describes a *transformation* (also known as a Karp-reduction or many-one reduction), which is a type of reduction. It has become common to use the term *reduction* in reference to the specific type defined here.

Definition 2.1.5 (reduction) A reduction from a problem Π to a problem Π' (symbolized $\Pi \leq \Pi'$) is an algorithm A such that

1. for any instance I of Π , $A(I) = I'$ is an instance of Π' .
2. any solution for I' can be used to obtain a solution for I .

3. reductions must preserve some specified property of the source problem (e.g. polynomial solvability).

A polynomial time reduction is one in which all computation specified in Definition 2.1.5 can be completed in time bounded by a polynomial function of $|I|$. An important property of reductions used in this thesis is that they are transitive relations. Relying on the closure of polynomials under composition, Karp used polynomial time reductions to show that if an NP-complete problem has a polynomial time algorithm, then all problems in NP have polynomial time algorithms.

In computer science, complexity theory is used mainly as a tool for obtaining (possibly conditional) lower bounds on the efficiency of solving difficult problems. In this capacity, the notion of completeness may be replaced by the notion of hardness. A problem is *hard* for a complexity class if all problems in that class reduce to it by a reduction that preserves the defining property of the class. For any complexity class, the type of reduction used to show hardness for the class depends on the defining properties of the class in question.

The situation uncovered by classical complexity theory is that if a problem can be shown to be NP-hard, then searching for a polynomial time solution will result in frustration: any attempt at a polynomial time solution of one NP-hard problem is an attempt for all NP-complete problems, of which many thousands have been well studied. Dealing with NP-hard problems requires more sophisticated techniques.

2.1.1 Parameterized complexity

Parameterized complexity grew out of the need to find exact solutions to NP-hard problems. Perhaps the first discussion of these was the recognition that certain NP-hard problems could be solved exactly by dynamic programming in *pseudo-polynomial* time².

Definition 2.1.6 (parameterized problem) A *parameterized problem* is specified by three pieces of information: the input, the question and those parameters that are designated as fixed.

In parameterized complexity there is a distinction between the parameters and the input size for a problem. Any combination of the parameters may be fixed, so a

²An early result of this sort is the polynomial time integer programming algorithm for a fixed number of variables or constraints [56]. The technique used to obtain this result has been formalized in parameterized complexity under the name ‘reduction to problem kernel’.

special notation is used to indicate the set of fixed parameters for a problem. The parameterized version is indicated by the symbols (parameters) appearing in brackets immediately after the name (or abbreviation) of the problem. For example, the problem of finding a size k vertex cover for fixed k is written VERTEX COVER(k). Note that when a problem only has a single natural parameter³ that may be fixed, the notation is redundant and, for such problems, will be omitted throughout this thesis.

In classical complexity theory, 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 function that is polynomial in the size of the input, but may be exponential in the value of the fixed parameters.

Definition 2.1.7 (fixed parameter tractable) A parameterized problem Π is *fixed parameter tractable* if there is an absolute constant c , a function $f : \mathbb{N} \rightarrow \mathbb{N}$ and a collection of procedures $\{A_k : k \in \mathbb{N}\}$ such that for each instance $\langle x, k \rangle$ of Π :

- (i) $A_k(\langle x, k \rangle) \in O(f(k)|x|^c)$,
- (ii) $\langle x, k \rangle \in \Pi$ iff $A_k(\langle x, k \rangle) = \text{true}$.

A problem that is fixed parameter tractable is said to reside in the parameterized complexity class FPT. The value of the parameterized point of view for hard problems becomes evident when exact algorithms are required. Parameterized complexity exploits combinatorial relationships that are particular to a subset of problem instances, while not sufficiently constraining to render the general problem *easy*. One point that is very important to keep in mind while considering parameterized complexity is the distinction between a parameter being *fixed*, and some value being *constant*. A constant may appear anywhere in the complexity expression of an algorithm. An example is $O(n^c)$ for constant c . If a value k is fixed, then k may only appear as an exponent if the corresponding base is a constant or another fixed parameter; e.g. $O(2^{f(k)}n^c)$ or $O(f(k)^{g(k)}n^c)$. The result of this distinction is that the complexity expression *depends only polynomially on the size of the input*.

The strength of parameterized complexity is in the general techniques it has produced for the design of efficient parameterized algorithms. These are outlined in

³The meaning of ‘natural parameter’ is subjective. For any abstract computational problem, what constitutes a parameter is likely to depend on how useful the abstraction remains after a particular value has been fixed.

[26] and include the methods of *bounded search tree*, *reduction to problem kernel*, and *perfect hashing*. For large classes of problems with natural graph-theoretic abstractions, the method of *finite obstruction sets* is another useful tool in the design of fixed parameter tractable algorithms.

Traditional intractability is associated with the conjecture that no polynomial time algorithm exists for any NP-complete problems. Although this has yet to be proved, the concept is very important to the study of algorithms. Parameterized complexity draws finer distinctions between problems and organizes them according to an infinite hierarchy of classes *within* the class NP.

Just as Cook's [21] characterization of NP is founded on the basic problem SAT, parameterized complexity has analogous problems to anchor its complexity classes. The central problem is called WEIGHTED CIRCUIT SATISFIABILITY (defined below) and can be thought of as providing a model for parameterized non-deterministic computation. The parameterized version of this problem defines the infinite hierarchy of parameterized complexity classes inside NP, although natural problems seem only to reside in five well-populated classes.

Definition 2.1.8 (Boolean circuit) 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 labeled from the set $\{0, 1, x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$. The nodes of fan-in greater than 0 are called *gates* and are labeled either AND, OR, 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 2.1.9 (weft) 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.

WEIGHTED WEFT t DEPTH h CIRCUIT SATISFIABILITY ($\text{WCS}_{t,h}$)

Instance: A weft t depth h decision circuit C .

Parameter: A positive integer k .

Question: Does C have a weight k satisfying assignment?

The set of problems reducible (under parameterized reductions, defined below) to $\text{WCS}_{t,h}$ for any h , forms the class called $\text{W}[t]$. When reducing a problem to $\text{WCS}_{t,h}$, it is sufficient to design a circuit that is satisfiable just in case the source problem has a positive solution.

Definition 2.1.10 For any t , the class $W[t]$ consists of all problems reducible, by a parameterized reduction, to $WCS_{t,h}$, where h is an arbitrary constant.

Definition 2.1.11 $W[P]$ is the class of all problems reducible, by a parameterized reduction, to WEIGHTED CIRCUIT SATISFIABILITY.

Definition 2.1.12 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 an arbitrary function independent of n .

The parameterized complexity classes are organized into the W -hierarchy:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq \cdots \subseteq XP$$

These classes are related to P and NP as:

$$P \subseteq FPT \subseteq \cdots \subseteq XP \subseteq NP$$

From a complexity theoretic point of view, parameterized complexity is based on the conjecture that $FPT \neq W[1]$. It is also believed that the other relationships in the hierarchy are proper [26].

Classification of problems according to the W -hierarchy makes use of parameterized reductions, which preserve membership in the classes.

Definition 2.1.13 (parameterized reduction) Let Π and Π' be two parameterized problems. A parameterized reduction from Π to Π' is an algorithm A that transforms an instance $\langle x, k \rangle$ of Π into an instance of $\langle x', k' \rangle$ of Π' such that:

1. A runs in time $O(f(k)|x|^c)$ for arbitrary function f (independent of $|x|$) and constant c (independent of both $|x|$ and k).
2. $k' = g(k)$ for some arbitrary function g independent of $|x|$.
3. $\langle x, k \rangle \in \Pi$ if and only if $\langle x', k' \rangle \in \Pi'$.

The central problem examined in this thesis has many aspects that can be exploited in the search for efficient fixed parameter algorithms. Such problems benefit from a uniform and systematic treatment.

Definition 2.1.14 (systematic parameterized complexity analysis [90]) Given a decision problem Π and some subset $S = \{s_1, \dots, s_n\}$ of the aspects of Π , a *systematic parameterized complexity analysis of Π relative to S* determines the parameterized complexity of Π relative to all $2^n - 1$ non-empty subsets of S .

The goal of systematic parameterized complexity analysis is the construction of a map that indicates the relation between sets of parameters and the complexity of a problem with respect to those parameters.

2.1.2 Approximation theory

Approximation theory is a more established tool for dealing with hard problems. The first systematic discussion appeared in [48] and was subsequently expanded in [36]. Much of approximation theory relies more on analysis than design in that many of the best approximation algorithms are very simple, but without a guarantee on their performance, using them is often unwise. As pointed out in [79], approximation is most useful when your input may not be an exact representation of the real problem being abstracted; in such cases, exact solutions to the computational problem are still approximations for the real-world problem.

When discussing approximation, in addition to algorithm efficiency, the solution correctness, or approximation quality, is also of concern. Various measures of approximation quality have been proposed. The two most important are the *relative error* and the *performance ratio*. The latter measure is used throughout this thesis.

Definition 2.1.15 (performance ratio) Let x be an instance of optimization problem Π having optimal solution $\text{opt}(x)$. Let A be an algorithm 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

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

Algorithm A is a ρ -approximation algorithm if and only if A always returns a solution with performance ratio less than or equal to ρ .

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 a function of $|x|$, some other parameter to the problem, or a constant. Unless otherwise stated, it is assumed that approximation algorithms are polynomial. Intuitively, the lower values of ρ are often achieved by sacrificing efficiency. This idea is formalized by the notion of an approximation scheme, the most important type being those that require polynomial time.

Definition 2.1.16 (polynomial time approximation scheme, PTAS) Let A_ρ (for rational number $\rho > 1$) be a family of approximation algorithms for optimization problem Π . If, for each fixed ρ , algorithm A_ρ is a ρ -approximation algorithm and its running time is polynomial in the size of the input, then A_ρ is called a polynomial time approximation scheme (PTAS). If, in addition, the running time of A_ρ is a polynomial function of $1/(\rho - 1)$, then A_ρ is called a fully-polynomial time approximation scheme (FPTAS).

A useful theory of complexity of approximation has been developed for optimization problems. The class of all optimization problems having corresponding decision versions in NP is called NPO. Within NPO, there are many classes defined by the approximability of their members. APX is the class of constant approximable problems, including VERTEX COVER and 3 SAT [45]. Additional classes have been defined between APX and NPO; these include the hierarchy of $f(n)$ -approximable problems such as log-APX and poly-APX. The classes PTAS and FPTAS contain those problems for which polynomial and fully polynomial time approximation schemes are known. The most useful classes are related as follows.

$$\text{FPTAS} \subseteq \text{PTAS} \subseteq \text{APX} \subseteq \text{NPO}.$$

The class relations are strict if, and only if, $\text{P} \neq \text{NP}$.

Reductions are also used to classify difficult optimization problems. The L -reduction [67] (linear reduction) requires that the source and target solution sizes, as well as their quality, be linearly related. This type of reduction was intended to show hardness and membership in the syntactic class MAX SNP of optimization problems. It was subsequently discovered that MAX SNP is exactly the class of constant factor approximable optimization problems, and that outside this class the L -reduction is of little use. As the structure of approximability was scrutinized during the 1990s, many (in fact more than a dozen) different types of approximability preserving reductions were proposed. The simplest of these reductions, first appearing in [4], is sufficient for the present purpose.

Definition 2.1.17 (gap preserving reduction, GP-reduction, \leq_{GP}) Let Π and Π' be two minimization problems. A gap-preserving reduction (GP-reduction, \leq_{GP}) from Π to Π' with parameters (c, ρ) and (c', ρ') is a polynomial-time algorithm f . For each instance x of Π , f produces an instance $x' = f(x)$ of Π' . The optima of x

and x' , say $\text{opt}(x)$ and $\text{opt}(x')$ respectively, satisfy the following properties:

$$\begin{aligned}\text{opt}(x) \leq c &\Rightarrow \text{opt}(x') \leq c', \\ \text{opt}(x) > c\rho &\Rightarrow \text{opt}(x') > c'\rho',\end{aligned}$$

where (c, ρ) and (c', ρ') are functions of $|x|$ and $|x'|$ respectively, and $\rho, \rho' \geq 1$.

Definition 2.1.17 specifically refers to minimization problems and can easily be adapted for maximization problems. Like other reductions, the usefulness of GP-reductions rests on a conjecture about the hardness of a particular problem, the transitivity of the reduction, and the closure of polynomials under composition. Although it is implied by the name, GP-reductions allow the gap to become larger or smaller, as long as some gap remains intact [4].

2.2 Molecular Sequence Patterns

This section is an introduction to the concepts of patterns in molecular sequences that motivate the research in this thesis. Many practical problems related to the discovery of patterns arise in molecular biology. Background information is given first, followed by a discussion of how patterns can be abstracted for computational processing.

2.2.1 Review of molecular biology

The information provided here is intended to familiarize the reader with those concepts sufficient to understand the discussion in later sections. For more information, the reader may consult the references [8, 81, 85].

Molecular genetics is primarily concerned with the relationship between information molecules (DNA and RNA) and working molecules (proteins). The basic relationship between DNA, RNA and protein is often referred to as the *central dogma of molecular biology*. The central dogma, put forth by Francis Crick [24], is a general theory. In biology all general theories have exceptions. The central dogma has survived as a matter of pragmatics; no other theory has better described the same concepts.

The central dogma holds that genetic information is encoded by patterns in DNA. These patterns are called genes. Through the process of transcription, the

information of a gene is converted into a messenger RNA molecule. When messenger RNA encounters ribosomes, the genetic information is used to produce a protein. In essence, genetic information within the cell flows from DNA to RNA to protein. When genetic information is transmitted from cell to cell, as happens when cells divide, the information flows from DNA to DNA through the process of DNA replication.

DNA (deoxyribonucleic acid) molecules are large polymers with a backbone of alternating sugar and phosphate residues [85]. A nitrogenous base is covalently bonded to the sugar residues. The four types of bases found in DNA are adenine (*a*), cytosine (*c*), guanine (*g*), and thymine (*t*). RNA (ribonucleic acid) molecules are similar chemically (with a different sugar) and have uracil (*u*) bases instead of thymine.

Proteins are composed of one or more polypeptide chains (throughout this thesis, the terms protein and polypeptide are used synonymously). The monomeric units making up a protein are amino acids. The amino acids are referred to as residues, of which there are twenty commonly occurring in nature. The sequence of amino acid residues that defines a particular protein is called its primary structure.

Though the chemical behavior of proteins is complex and remains mysterious [33], that of nucleic acid molecules is well understood. The important chemical properties of nucleic acids are that hydrogen bonds form between bases according to the Watson-Crick rules [93]: *a* bonds with *t* (*u* for RNA), and *c* bonds with *g*. These pairs are called canonical base pairs, and form when members of a pair are placed opposite each other (*i.e.* in different sequences or sufficiently distant in the same sequence). Base pairs are also called *complementary* bases, and this concept extends in a natural way to sequences of bases. When two segments of DNA have a sufficient number of binding base pairs, the segments are said to *hybridize*. Thus any two segments composed entirely of complementary bases are said to be complementary segments. For instance the (ordered) segments *acagt* and *tgatc* are complementary, since the base at any position in the first segment is complementary to the base at that same position in the second segment. Hybridization behavior is central to the *in vivo* function of DNA and RNA; it is also exploited by many important *in vitro* (and more recently *in silico*) biological techniques.

2.2.2 The nature of sequence patterns

There is no disputing the claim that important information is contained in the patterns of biological macromolecules. Unfortunately, there is no general characterization for the encoding of information in bio-molecular sequences. Different types of information are necessarily represented differently.

Of the important patterns in bio-molecular sequences, the most famous is the gene. In a fixed cellular environment the base sequence of a gene completely determines the conformation of the encoded protein. The encoding scheme used in genes is called the genetic code and maps triples of bases to amino acids in the resulting protein (note that the genetic code is surjective but not injective). In this encoding, the contribution of a single base to the shape of the encoded protein cannot be determined without knowledge of the adjacent bases, and the location of the base in the reading frame. When a gene is modified, the result may or may not be a conformational change in the encoded protein, the effect of which is usually realized at a great (cellular) distance from the gene.

When a gene is transcribed and the encoded protein is eventually built, the gene is said to have been expressed (although gene expression is often a matter of degrees, and biologists never treat expression as a discrete phenomenon). The expression of a gene depends on other patterns in DNA, such as regulatory protein binding sites. Genes are much longer than binding sites; the patterns therefore differ in a physical characteristic. The two types of patterns also differ qualitatively. The binding site encodes the important information of what other molecules will bind to it chemically. This binding property is physical and acts immediately both in terms of time and space, with information being encoded by single bases instead of triplets. The total amount of “binding” is equal to the sum of the contributions of each base making up the binding site, and therefore the contribution of each individual base can be predicted without knowing the identity of the adjacent bases.

A third type of pattern with important differences is the restriction site. Restriction sites are segments of DNA recognized by restriction enzymes. The restriction enzymes cut the DNA sequence at the restriction sites, a process that plays a role in gene expression as well as cell defense. The most interesting property of restriction sites is that they tend to be complemented palindromes, which are DNA segments for which the second half of the segment becomes identical to the first half when reversed and complemented (according to the Watson-Crick rules). Similar to the binding site patterns, the property of being a complemented palindrome is essential to binding activity immediately at the site of the pattern. The palindromic encoding,

however, is quite different from the genetic code. The genetic code can be expressed as a regular language, while palindromes require context free representation.

The above three examples illustrate the difficulty of obtaining a general characterization for information in DNA. This problem is also seen in RNA and proteins. It is unreasonable to expect any characterization to capture the properties of all classes of sequence patterns. Specific biological knowledge is still required to assess the importance of a putative pattern, creating a need for automated pattern finding methods that are fast enough to keep pace with the biologist's intuition.

2.2.3 Operational definition of “pattern”

Patterns in bio-molecular sequences can loosely be defined as any feature that repeats. This is much too broad a description so an operational definition must be formulated to restrict the class of objects under discussion. The patterns that have been the focus of attention in computational biology can be distinguished on many dimensions and these must be addressed in any operational definition.

In 1989, Staden [82] described a hierarchy of nine classes of motifs, and defined a pattern as “a higher order of structure formed of motifs”. Common to all classes of motifs was the property of being “a contiguous section of a sequence”. For any sequence, the number of occurrences of any contiguous patterns is bounded by a polynomial (more precisely, a quadratic) function of the length of the sequence, while the number of non-contiguous patterns is exponential. Non-contiguous patterns offer a more expressive representation, but these patterns are also more difficult to manipulate from an algorithmic point of view. Staden's distinction between “pattern” and “motif” introduces the notion of a *meta-pattern*, which will not be treated here. It is important to acknowledge that making a commitment along this dimension forces a compromise. In order to allow specific properties of patterns to be exploited in computation, we restrict the relevance of our methods to specific types of biologically meaningful patterns.

Some means of comparing patterns is essential to our operational definition. Two distance measures that are commonly used to compare sequences are the Levenshtein and Hamming distances. The Levenshtein distance (also called edit distance) is the minimum number of insertions, deletions or substitutions that are needed to transform one string into another. This distance does not imply that patterns are non-contiguous, indeed there are many types of Levenshtein distances [44]. These are useful when comparing two strings of differing length or strings for which the differences are in fact caused by edit operations. Hamming distance

[17] originated in coding theory and is the minimum number of substitutions that are needed to transform one string into another. This distance is more restrictive, but does correctly model a large class of biological objects. Hamming distance can be made more general by the addition of wild-cards and meta-symbols that can represent different monomers (basic units). Both of these distances satisfy the axioms of a metric.

There are biological issues to consider when comparing these two common distances. The motivation for using Levenshtein distance is often its utility when modeling evolution. When mutation is at action, sequences evolve through the primitive operations of substitution, insertion and deletion⁴. When comparing sequences to test a hypothesis based on evolutionary relationships, Levenshtein distance is valuable. Relationships between sequences have practical value beyond the evolutionary information. When oligonucleotides are designed to hybridize (*e.g.* probes and primers, described below), insertions and deletions can safely be ignored and Hamming distance becomes more useful. The destabilizing effect of one mismatch between two nearly complementary segments provides an illustration. The free energy caused by a substitution is approximately 0.8 kcal/mol. When insertions are allowed, the free energy added by a gap is 3.3 kcal/mol [54]. This difference is the result of inevitable steric tension (bending the hydrocarbon backbone) caused by the gap.

A different way to view a pattern is as its set of *occurrences*. An occurrence of a pattern is any string fitting the criteria specified by the pattern. This notion is meaningless in the case of exactly matching patterns, for which any two occurrence are identical; exactly matching patterns are not of concern in this thesis. Among inexact (or approximate) patterns, there is a distinction between deterministic patterns and probabilistic patterns [16]. With respect to deterministic patterns all occurrences are equal, meaning the property of being an occurrence is boolean. This remains true no matter how the deterministic pattern is specified. The notion of a probabilistic pattern is different in that it imposes a partial order on its occurrences. For these, being an occurrence is a matter of degrees. Some common representations for probabilistic patterns are hidden Markov models [53], neural nets [95] and (most importantly) profiles [42], although each of these can be made deterministic simply by introducing a threshold. Other representations, such as regular expressions or signatures [44], are more commonly associated with the deterministic variety. We

⁴There are other mechanisms acting on a large scale such as inversion, translocation, transposition and duplication [51]. The patterns discussed in this thesis are small enough not to be significantly affected by these other mechanisms.

remark that for theoretical analysis deterministic patterns are more convenient. In biological applications, probabilistic patterns are often more useful because they provide an objective function of higher resolution that can better distinguish between pattern occurrences.

Being aware of how patterns might be characterized is essential, but ultimately the decision depends on the types of biological objects that the definition must admit. Single amino acid residues are important for the action of many proteins, and these can be legitimate patterns, as are entire genomes.

The operational definition is now formalized. Pattern and motif are assumed synonymous. For the present discussion, a pattern is a contiguous string of specified length over a finite alphabet that is identical to the alphabet of any occurrences. Patterns are also deterministic, and are compared using the Hamming distance. These patterns generally do not contain gaps or wild-cards, and may only be compared if similar in length. Patterns are represented as strings over finite alphabets that map bijectively to some set of monomeric units making up a molecular sequence. Because we are only concerned with contiguous patterns, we will not require the usual distinction between a sequence and a string. Here the term sequence is used because the strings under discussion are usually representing molecular sequences. Alphabets usually represent the set of nucleic acids or the set of amino acids, but unless otherwise stated, the nucleic acids of DNA are assumed. An occurrence of a pattern is a string of the same length as the pattern that is within some specified Hamming distance of the pattern.

The operational definition here will admit short sequences, on the order of 10^1 to 10^2 units in length. These patterns are capable of describing DNA binding sites, genes, amino acid motifs and polypeptides. The definition encompasses many of the types of patterns of interest in automatic discovery. Examples of these are given in Section 2.2.4.

2.2.4 Importance of sequence pattern discovery

Applications of pattern discovery in the biological sciences are numerous and the examples given here are not comprehensive. All the examples presented below have either explicitly or implicitly influenced the material presented in later chapters.

Classification One of the most obvious uses of pattern discovery in biological sequences is sequence classification. The goal of such classification is, given a set of sequences, find patterns within the sequences that can be used to define member-

ship in the set. This is the basic problem of inductive learning: given a training set of sequences, find the pattern that will allow the classification of sequences not in the training set. The PROSITE database [7] is an example of this use of patterns. Although classification is not directly motivating the problems examined in this thesis, its influence is significant in that many important techniques of pattern finding (some influencing the present research) have been concerned with classification problems.

Function Prediction Another strong motivating factor is homology (*i.e.* structural similarity) based function prediction. This method uses the homology between an unknown sequence and a sequence of known function to extrapolate the function of the unknown sequence. The method is based on Gusfield's first fact of biological sequence analysis:

“In bio-molecular sequences (DNA, RNA, or amino acid sequences), high sequence similarity usually implies significant functional or structural similarity.”

– Gusfield [44]

So if two sequences share a significant homology, they probably have similar functions. This fact is not absolute (recall that in biology there are exceptions to every rule), but it provides a basis for inquiry that can generate inferences about function. It is also important to note that the converse does not hold. These inferences can then be verified using traditional biological and chemical methods.

One successful example of function prediction has been the classification of *E. coli* genes according to a pattern appearing upstream of the gene. Cyclic AMP (3', 5'-cyclic adenosine monophosphate) is a regulatory element in animal cells and bacteria. It is known that CRP (Cyclic AMP response protein) works in conjunction with Cyclic AMP to regulate the lactose operon in *E. coli* [11]. The binding specificity of CRP is well known [10] and sequences containing CRP binding sites have been used extensively as test sequences for pattern finding programs [55, 71, 83]. Attempts to characterize the CRP binding site in terms of a pattern have also motivated the development of statistical scoring schemes.

Oligonucleotide Design The design of custom oligonucleotides, or oligos, short segments with particular hybridization specificity, has also motivated the development of pattern finding methods. Such custom oligos as probes and primers have become essential to research in the biological sciences.

- *Polymerase Chain Reaction* (PCR) is an established experimental technique that allows the amplification of a DNA segment (called a template) between two known DNA segments [64]. This *in vitro* technique makes use of the natural machinery of DNA replication. PCR amplification is achieved by using oligonucleotide primers, which are single stranded DNA segments complementary (with the possibility of a small number of mismatches) to the known segments bracketing the template. As explained in Section 2.2.1, it is the nature of DNA for complementary strands to hybridize by forming hydrogen bonds between complementary bases. A solution is created containing the target DNA (the sequence containing the template). The primers are inserted into the solution and hybridize to the bracketing segments, single stranded due to heat denaturation, then extended as *in vivo* by DNA polymerase. The procedure is cycled through *generations* where the number of sequences of the template doubles each generation.

The specificity of amplification by PCR depends on the specificity of primer hybridization. It is essential to design primers that only hybridize with the segments bracketing the template. In case there are multiple targets, there is a need to design primers that may be able to recognize multiple segments. For primers with multiple targets, the maximum number of mismatches that still allow hybridization is a significant factor in primer design.

- Another experimental technique that requires specially designed oligonucleotides is the use of *probes* for identifying molecular sequences within the cell or *in vitro* [84]. Chemically modified segments of DNA or RNA can be inserted into cells or solution and their binding behavior observed. Observation of the sequences is facilitated by the chemical modification. The probes are designed to hybridize with specific types of sequences. This has numerous applications, some of the most interesting include forensics, genetic screening and diagnosis. For example, segments of DNA can be constructed with sequences complementary to bacterial segments suspected of causing an infection. The probes are inserted into an infected cell. If the probes hybridize to sequences inside the cell it can be concluded that the bacterial sequence is present. Such a conclusion relies heavily on the ability of the probe to identify its target within a complex mixture of biological macromolecules.

More complex hybridization behavior is desired when operating under uncertain conditions. If a probe is developed from a known polypeptide, or intended to hybridize with all members of a particular *family* of segments,

then the pattern representing the probe will not be exact (*i.e.* the pattern will have multiple distinct occurrences). The purpose of complex and degenerate probes is not to target unambiguous sequences, but to detect a set of segments that can be examined more precisely later.

- One of the most interesting applications of custom oligonucleotides is antisense technology. Antisense technology is the name given to a class of technologies related by their use of antisense oligonucleotides to explore and control gene expression [65]. These techniques have shown promise as therapeutic agents for the treatment of HIV and cancer [43].

The name *antisense* refers to the strand that is complementary to the strand of interest, and the hybridization of complementary strands is the foundation of antisense technology. It is thought that it will provide a general means of controlling gene expression [66]. One mechanism of action for antisense oligonucleotides, once introduced into the cell, is to hybridize with single stranded mRNA, preventing its translation. Another mechanism is to introduce the antisense strand into the nucleus having it form a triple helix with the promoter region of genes to be inhibited, interfering with normal transcription.

The implications of this technology are immense, and the technology is still hindered by issues relating to the delivery and degradation of the antisense oligonucleotide. Once these problems are solved, the challenge will be to design oligonucleotides with the proper hybridization properties. This will reduce to a pattern finding problem in sets of sequences.

2.3 General Properties of the Central Problem

The next three chapters contain a detailed theoretical analysis of the pattern discovery problem. The problem is abstracted into three distinct forms for the analysis, but the set of problem instances for each abstraction is essentially the same. This section introduces the decision problem investigated in Chapter 3 as a means of introducing some general properties that will be used throughout the remainder of this thesis.

COMMON APPROXIMATE SUBSTRING (CAS)

Instance: A family $\mathcal{F} = \{S_1, \dots, S_m\}$ of strings over an alphabet Σ such that for

all $S \in \mathcal{F}$, $|S| = n$, and two positive integers l and d such that $1 \leq d < l \leq n$.

Question: Is there a string $C \in \Sigma^l$ such that for each $S \in \mathcal{F}$, there exists a length l substring s of S with the property that $d_H(s, C) \leq d$?

The function d_H denotes the Hamming distance between strings. The following definition places the problem in a geometric context.

Definition 2.3.1 (radius) Let \mathcal{F} be a family of strings, and l a positive integer. For any $S_i \in \mathcal{F}$, let s_{ij} be the j^{th} length l substring of S_i . The radius of \mathcal{F} is defined as

$$\text{radius}(\mathcal{F}) = \min_{C \in \Sigma^l} \max_{S_i \in \mathcal{F}} \min_{1 \leq j \leq m-l+1} d_H(s_{ij}, C).$$

For any $C \in \Sigma^l$, the radius of C with respect to \mathcal{F} is defined as

$$\text{radius}_{\mathcal{F}}(C) = \max_{S_i \in \mathcal{F}} \min_{1 \leq j \leq m-l+1} d_H(s_{ij}, C).$$

The next results are essential to understanding the nature of the problem instances. The proofs are omitted.

Proposition 2.1 *Let \mathcal{F} be a family of strings with $\text{radius}(\mathcal{F}) = d$, and let $C \in \Sigma^l$ satisfy $\text{radius}_{\mathcal{F}}(C) = d$. Consider any two strings $S, S' \in \mathcal{F}$ with length l substrings s and s' , respectively. If s and s' are occurrences of C , then*

$$d_H(s, s') \leq 2(\text{radius}(\mathcal{F})).$$

If, in addition, $d_H(s, s')$ is the largest distance between any pair of occurrences of C , then

$$\text{radius}(\mathcal{F}) \leq d_H(s, s').$$

For any string S , let $S[p]$ denote the character in the p^{th} position of S . Let s_1 and s_2 be two strings in Σ^l . Strings s_1 and s_2 are said to *agree* at position p if and only if $s_1[p] = s_2[p]$, otherwise s_1 and s_2 disagree on position p .

Proposition 2.2 *Let \mathcal{F} be a family of strings with $\text{radius}(\mathcal{F}) = d$, and let $C \in \Sigma^l$ satisfy $\text{radius}_{\mathcal{F}}(C) = d$. If $\{s_1, \dots, s_m\}$ is a set of occurrences, one from each $S_i \in \mathcal{F}$, and $1 \leq p \leq l$, then for any character α ,*

$$(\forall 1 \leq i \leq m, s_i[p] = \alpha) \Rightarrow (C[p] = \alpha),$$

and there exists a $C' \in \Sigma^l$ such that

$$(C'[p] = \alpha) \Rightarrow (\exists 1 \leq i \leq m, s_i[p] = \alpha).$$

Definition 2.3.2 (neighborhood) For a string $S \in \Sigma^n$, with $n \geq l$, the (l, d) -neighborhood of S is the set

$$\{s' : s' \in \Sigma^l \wedge d_H(s, s') \leq d \text{ for some substring } s \text{ of } S \text{ with } |s| = l\}.$$

For any string S , we use $N_{l,d}(S)$ to denote the (l, d) -neighborhood of S . For a family \mathcal{F} of strings, the (l, d) -neighborhood of \mathcal{F} is the set

$$\{s' : s' \in \Sigma^l \wedge \forall S \in \mathcal{F}, s' \in N_{l,d}(S)\},$$

and is denoted $N_{l,d}(\mathcal{F})$.

We also define the value

$$N = \sum_{i=0}^d \binom{l}{i} (|\Sigma| - 1)^i,$$

and note that this value appears throughout our analysis. The significance of N is that for a string s with $|s| = l$, $N = |N_{l,d}(s)|$.

Definition 2.3.3 (column majority) For any set of strings $K \subseteq \Sigma^l$ and any position p ($1 \leq p \leq l$), if α satisfies

$$|\{s \in K : s[p] = \alpha\}| = \max_{\beta \in \Sigma} |\{s \in K : s[p] = \beta\}|,$$

then α is a column majority character for position p in K . A column majority string is defined in the natural way as a string having a column majority character at each position.

Note that column majority characters and strings may not be unique, and there can be up to $|\Sigma|$ column majority characters at any position, and up to $|\Sigma|^l$ distinct column majority strings.

The COMMON APPROXIMATE SUBSTRING problem does not immediately suggest a polynomial time solution. In such cases the logical first step in complexity analysis is to determine membership in NP. The algorithm VALID (presented in Algorithm 1) demonstrates that COMMON APPROXIMATE SUBSTRING can be solved in non-deterministic polynomial time. The algorithm simply aligns the putative center with all possible length l substrings in the set of sequences given as input, counts the number of matching positions, and returns whether or not each sequence contained an occurrence.

Algorithm 1: Pseudocode for the VALID algorithm.

Input: A family of strings $\mathcal{F} = \{S_1, \dots, S_m\}$, a string $\mathcal{C} \in \Sigma^l$, and an integer d such that $1 \leq d \leq l$.

Output: A boolean value indicating whether there is a length l substring in each member of \mathcal{F} with distance at most d from \mathcal{C} .

VALID($\mathcal{C}, \mathcal{F}, d$)

1. $l \leftarrow |\mathcal{C}|$
2. $x \leftarrow \text{true}$
3. **for** $i \leftarrow 1$ **to** m
4. $x_i \leftarrow \text{false}$
5. **for each** length l substring s of S_i
6. **if** $d_H(\mathcal{C}, s) \leq d$ **then** $x_i \leftarrow \text{true}$
7. **if** $x_i = \text{false}$ **then** $x \leftarrow \text{false}$
8. **return** x

The VALID procedure can non-deterministically solve COMMON APPROXIMATE SUBSTRING if it is coupled with an oracle for producing a valid center whenever one exists. This algorithm is an important subroutine in many procedures throughout this thesis. The time complexity of the algorithm as it is presented in Algorithm 1 is $\Theta(mnl)$. A similar procedure is needed if the radius of a problem instance is required. Such a procedure is useful in the context of optimization. Pseudocode for SCORE is provided in Algorithm 2. The SCORE algorithm has the same complexity as the VALID algorithm, which is apparent from the similar structure of the algorithms.

Algorithm 2: Pseudocode for the SCORE algorithm.

Input: A family of strings $\mathcal{F} = \{S_1, \dots, S_m\}$ and a string $\mathcal{C} \in \Sigma^l$.

Output: The radius of \mathcal{C} with respect to \mathcal{F} .

SCORE(\mathcal{C}, \mathcal{F})

1. $l \leftarrow |\mathcal{C}|$
2. $d \leftarrow 0$
3. **for** $i \leftarrow 1$ **to** m
4. $d_i \leftarrow \infty$
5. **for each** length l substring s of S_i
6. **if** $d_H(\mathcal{C}, s) \leq d_i$ **then** $d_i \leftarrow d_H(\mathcal{C}, s)$
7. **if** $d_i > d$ **then** $d \leftarrow d_i$
8. **return** d

After determining membership in NP, the crucial question is whether COMMON APPROXIMATE SUBSTRING is in P, or NP-complete. Unfortunately, the problem is known to be NP-complete, a fact first proved in [34] for the restricted case where $\Sigma = \{0, 1\}$ and $l = n$. The proof is not presented here; various proofs of the NP-hardness of COMMON APPROXIMATE SUBSTRING appear throughout Chapters 3 and 4.

Chapter 3

Parameterized Complexity Analysis

For the general parameterized problem COMMON APPROXIMATE SUBSTRING there are many parameters that can be fixed making it a good candidate for study under parameterized complexity. The parameterized version is stated as:

COMMON APPROXIMATE SUBSTRING, CAS

Instance: A set $\mathcal{F} = \{S_1, \dots, S_m\}$ of strings over an alphabet Σ such that $|S_i| \leq n$, $1 \leq i \leq m$, and positive integers l and d such that $1 \leq d \leq l \leq n$.

Parameter: Alphabet Σ and positive integers m, n, l and d .

Question: Is there a string $C \in \Sigma^l$ such that for each string $S \in \mathcal{F}$, C is Hamming distance $\leq d$ from some length- l substring of \mathcal{F} ?

The size of the input is mn , which is often dominated by the length of the strings in \mathcal{F} , but occasionally $m \gg n$ (see, e.g., [86]). For this reason we include n in the list of possible parameters.

3.1 Fixed Parameter Tractable Variants

Exact FPT algorithms for COMMON APPROXIMATE SUBSTRING are of particular interest in computational biology [14]. The discussion in Section 2.2.4 described many examples where the values of m, l, d and Σ are very small compared to n . Among them were the design of DNA sequence primers, probes to detect sequence presence and distinguish sequences, and complementary sequences to block binding sites. Typically these require only small parameter values. For example, instances of COMMON APPROXIMATE SUBSTRING that occur in the design of DNA primers for groups of sequences have very small values for $|\Sigma|, d$, and l , e.g., $|\Sigma| = 4, d \leq 3$, and $l \leq 25$ [30]. COMMON APPROXIMATE SUBSTRING has a substantial body of previous algorithmic work, but much of this work is on heuristic algorithms. Theorem 3.2 also appears in [29] where CAS is treated in the context

of a more general problem.

Simple enumeration Generate all possible strings of length l over Σ and examine each of these strings to see if it is a center for \mathcal{F} . There are $|\Sigma|^l$ such strings and each of these strings can be checked in $O(mnl)$ time using the VALID algorithm from Section 2.3; hence, the algorithm as a whole runs in $O(|\Sigma|^l mnl)$ time and $O(mn)$ space. This is essentially the first algorithm given in [92]. The advantage of this approach is that the total space required is only a constant function of the input size.

The DevelopCenter algorithm We introduce a new character $x \notin \Sigma$, called the *blocking character*. The importance of x is that it always induces a mismatch when compared to a character in \mathcal{F} . Let \mathcal{C} be a length l string containing at most d occurrences of the blocking character x . Let s be a length l substring of string $S \in \mathcal{F}$. A substitution of \mathcal{C} under s is a set of modifications to \mathcal{C} that replaces of a subset of the occurrences of character x in \mathcal{C} with the characters appearing in corresponding positions of s . A *minimal matching* substitution is a substitution that results in $d_H(\mathcal{C}, s) \leq d$, with the additional property that no substitution replacing a *subset of those same positions* results in $d_H(\mathcal{C}, s) \leq d$ (note that a minimal matching substitution need not be unique). As an example, consider the strings $acgta$ and $axxxa$. If $axxxa$ is to be modified so that it is a center for $acgta$ with maximum distance 1, then there are three minimal matching substitutions. When applied to $axxxa$, the minimal matching substitutions produce the set of strings $\{acgxa, axgta, acxta\}$.

The DEVELOPCENTER algorithm is based on the observation that to find a center, it is sufficient to obtain a single occurrence of the center, then change characters in up to d positions of that occurrence. When the algorithm begins, an arbitrary string $S \in \mathcal{F}$ is removed from \mathcal{F} . For each length l substring s of S , let $\mathcal{C} = s$, and for each size d set of positions in \mathcal{C} , change the characters at those positions to the blocking character x . The second stage of the algorithm proceeds recursively, developing a center \mathcal{C} by substituting characters of Σ back into positions occupied by the blocking character. For each recursive call, a string S' is arbitrarily removed from \mathcal{F} . If the center \mathcal{C} currently being developed has distance at most d from some substring of S' , then another recursive call is made immediately. If all substrings of S' differ from \mathcal{C} in more than d positions, alternative centers are produced by modifying \mathcal{C} . For each substring $s \in S'$, such that $d_H(s, \mathcal{C}) \leq 2d$, a set M of alternative centers is obtained by making minimal matching substitution to \mathcal{C} under s . For each

\mathcal{C}' in M , a recursive call is made, with \mathcal{C}' passed as the center to further develop. If the set \mathcal{F} becomes empty, then the center currently being developed is valid for the *original* set of strings \mathcal{F} (i.e. before any strings were removed), and the algorithm returns that center.

The set M of alternative centers is defined as the set $mm(\mathcal{C}, s)$ of all strings \mathcal{C}' such that $d_H(s, \mathcal{C}') = d$, $d_H(\mathcal{C}', \mathcal{C}) = d_H(s, \mathcal{C}) - d$, and for all $1 \leq p \leq l$, $\mathcal{C}'[p] \neq \mathcal{C}[p]$ implies that $\mathcal{C}[p] = x$. The set $mm(\mathcal{C}, s)$ contains all strings obtained by making a minimal matching substitution to \mathcal{C} under s . Since all minimal matching substitutions make the same number $d' = d_H(s, \mathcal{C}) - d$ of changes to blocking characters in \mathcal{C} ,

$$|mm(\mathcal{C}, s)| = \binom{d}{d'} \leq \binom{d}{\frac{d}{2}}$$

and the set $mm(\mathcal{C}, s)$ can be computed in $O(\binom{d}{d/2})$ time.

The DEVELOPCENTER algorithm, described below in pseudocode, accepts as input a family of strings $\mathcal{F} = \{S_1, \dots, S_m\}$ and a string \mathcal{C} . For the initial call to DEVELOPCENTER, the string \mathcal{C} given as input is the empty string λ . The algorithm outputs a center for \mathcal{F} , if one exists.

Algorithm 3: Pseudocode for the DEVELOPCENTER algorithm.

DEVELOPCENTER(\mathcal{F}, \mathcal{C})

1. **if** $\mathcal{F} = \emptyset$ **then return** \mathcal{C}
2. **if** $\mathcal{C} = \lambda$
3. Let S be an arbitrary string in \mathcal{F} .
4. **for each** length l substring s of S
5. $\mathcal{C} \leftarrow s$
6. **for each** $B \subseteq \{1, \dots, l\}$, such that $|B| = d$
7. \forall positions $p \in B$, substitute $\mathcal{C}[p] \leftarrow x$
8. DEVELOPCENTER($\mathcal{F} \setminus \{S\}, \mathcal{C}$)
9. **if** $\mathcal{C} \neq \lambda$
10. Let S be an arbitrary string in \mathcal{F} .
11. branch \leftarrow true
12. **for each** length l substring s of S
13. **if** $d_H(s, \mathcal{C}) \leq d$ **then** branch \leftarrow false
14. **if** branch = false **then** DEVELOPCENTER($\mathcal{F} \setminus \{S\}, \mathcal{C}$)
15. **if** branch = true
16. **for each** length l substring s of S
17. **if** $d_H(s, \mathcal{C}) \leq 2d$
18. Let M be the set $mm(\mathcal{C}, s)$.
19. **for each** $\mathcal{C}' \in M$
20. DEVELOPCENTER($\mathcal{F} \setminus \{S\}, \mathcal{C}'$)

Theorem 3.1 *The DEVELOPCENTER algorithm requires $O(mn^{d+1}l^d2^{d^2})$ time.*

Proof. Consider the recursion tree of DEVELOPCENTER as the search space of the algorithm. The time complexity of the algorithm has a factor of $n \binom{l}{d}$ representing the out degree of the root of the search tree. A *branch point* refers to a string that the center must accommodate through a substitution. Since there can be at most d substitutions for blocking characters in a string, there are at most d branch points on any path from root to leaf in the search space. The out degree at each branch point is at most $n \binom{d}{d/2}$, corresponding to the maximum number of substrings that must be tried, multiplied by the maximum number of minimal matching substitutions that must be tried. The number of leaves in the search space cannot exceed $n \binom{l}{d} (n \binom{d}{d/2})^d$ and $\Theta(nm)$ time is required for each leaf. Hence, with the binomails in closed form, the time complexity of the algorithm is bounded by $O(mn^{d+1}l^d2^{d^2})$.

□

Of note is the absence of any parameter representing the alphabet, *i.e.* $|\Sigma|$ or both m and n , in the exponent of the time complexity expression for DEVELOP-CENTER.

Theorem 3.2 $CAS(|\Sigma|, l)$ and $CAS(n)$ are fixed parameter tractable.

As previously stated, there are problem instances where n is fixed [86]. For our purposes, $CAS(|\Sigma|, l)$ is the only tractable variant of CAS.

3.2 Parameterized Hardness

The hardness of several fixed-parameter variants of COMMON APPROXIMATE SUBSTRING is proved here by parameterized reductions from problems with known parameterized complexity. The following problems will serve as source problems in our reductions:

CLIQUE [36, Problem GT19]

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Is there a set $V' \subseteq V$ of k vertices that is a *clique* of G (that is, a set V' that forms a complete subgraph of G)?

DOMINATING CLIQUE [26, Page 463]

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Is there a set $V' \subseteq V$ of k vertices that is both a clique and a *dominating set* of G (that is, a set V' such that each vertex in G is either in V' or adjacent to a vertex in V')?

SET COVER [36, Problem SP5]

Instance: A set \mathcal{B} of elements, a family of sets \mathcal{L} such that $\mathcal{L}_i \subseteq \mathcal{B}$, ($1 \leq i \leq |\mathcal{L}|$) and a positive integer k .

Parameter: A positive integer k .

Question: Is there a size k subset $R \subseteq \mathcal{L}$ such that $\cup_{R_j \in R} R_j = \mathcal{B}$?

3.2.1 W[1]-hardness of CAS(m, l)

To show W[1]-hardness for CAS(m, l), we give a parameterized reduction from the W[1]-complete problem CLIQUE [26]. Note that versions of this reduction were developed independently in [29] and [32]. Let $G = (V, E)$ be a graph for which we wish to determine whether G has a k -clique. We construct a family \mathcal{F} of $m = f_1(k)$ strings over alphabet Σ that has a center of length $l = f_2(k)$ if and only if G contains a k -clique. Assume for convenience that the vertex set of G is $V = \{1, \dots, |V|\}$.

Target parameters The number of strings in \mathcal{F} is $m = f_1(k) = \binom{k}{2}$. The length of center \mathcal{C} is $l = f_2(k) = k + 2$, and the maximum distance between occurrence and center is $d = f_3(k) = k - 2$. The maximum length of any string in \mathcal{F} (which is not fixed in the reduction) is $n = f_4(G, k) = (2k + 4)(|E|)$.

The alphabet The string alphabet is $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$. We refer to these as vertex characters (Σ_1), unique characters (Σ_2), and alignment characters (Σ_3):

$$\begin{aligned}\Sigma_1 &= \{1, \dots, |V|\}, \\ \Sigma_2 &= \{\text{Set of characters occurring uniquely in } \mathcal{F}\}, \\ \Sigma_3 &= \{A, B\}.\end{aligned}$$

The characters of Σ_2 are denoted by u . All occurrences of this character are unique characters.

Substring gadgets We next describe the two ‘‘high level’’ component substrings used in the construction.

Edge Selectors:

$$\langle \text{edge}(i, j)(p, q) \rangle = Au^{(i-1)}pu^{(j-i-1)}qu^{(k-j)}B$$

Separators:

$$\langle \text{separator} \rangle = u^{k+2}$$

The reduction The $\binom{k}{2}$ strings in \mathcal{F} correspond to the $\binom{k}{2}$ edges in a k -clique:

$$\mathcal{F} = \{S_{ij} : 1 \leq i < j \leq k\}.$$

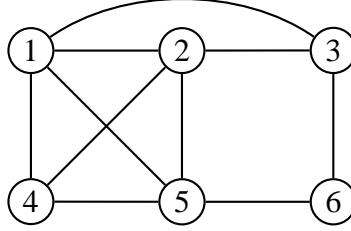


Figure 3.1: This is example graph 1. The vertex set $\{1, 2, 4, 5\}$ forms a 4-clique.

$$\begin{aligned}
S_{12}: & A12uuBu^6A13uuBu^6A14uuBu^6A23uuBu^6A24uuB \\
S_{13}: & A1u3uBu^6A1u4uBu^6A1u5uBu^6A2u4uBu^6A2u5uB \\
S_{14}: & A1uu4Bu^6A1uu5Bu^6A2uu5Bu^6A3uu6B \\
S_{23}: & Au23uBu^6Au24uBu^6Au25uBu^6Au45uB \\
S_{24}: & Au2u4Bu^6Au2u5Bu^6Au3u6B \\
S_{34}: & Auu36Bu^6Auu45Bu^6Auu56B
\end{aligned}$$

Figure 3.2: CAS(m, l) representation for example graph 1 (desired clique size $k = 4$).

String S_{ij} is composed of edge components arranged in the following manner (where product notation refers to concatenation and is ordered):

$$S_{ij} = \prod_{\substack{(p,q) \in E \\ p < q, i \leq p \\ j - i \leq q - p \\ q \leq |V| - k + j}} \langle \text{edge}(i, j)(p, q) \rangle \langle \text{separator} \rangle.$$

An example of the reduction for the graph in Figure 3.2.1 and a desired clique size 4 can be seen in Figure 3.2.1. Any center for \mathcal{F} will have the property that all positions other than the terminal positions will be occupied by vertex characters in ascending order.

The proof of correctness of this reduction exploits some interesting properties of \mathcal{F} . These properties are examined with the aid of the following definitions and conventions. An occurrence that begins and ends with alignment characters is said to be *in-phase*. *Vertex positions* are those positions in a string or substring occupied by characters from Σ_1 (the vertex characters). Note that for string S_{ij} , the vertex positions are positions i and j to the right of the initial alignment character. For any

vertex position i , the *vertex group* of i , denoted \mathcal{V}_i , is defined as the set

$$\mathcal{V}_i = \{S_{ix} : i < x \leq k\} \cup \{S_{xi} : 1 \leq x < i\}.$$

The intended role of \mathcal{V}_i is that the occurrences of center \mathcal{C} from \mathcal{V}_i determine the character at position i in \mathcal{C} . Without loss of generality, it is assumed that no two occurrences can come from the same string.

Lemma 3.1 *Let \mathcal{C} be a center for \mathcal{F} . The following are true:*

1. \mathcal{C} begins with character A and ends with character B .
2. No position in \mathcal{C} is occupied by a character from Σ_2 .
3. If \mathcal{I} is an occurrence of \mathcal{C} , then \mathcal{I} is in-phase.
4. The $k - 1$ occurrences from any vertex group are sufficient to completely determine \mathcal{C} .

Proof.

(1) Suppose \mathcal{C} begins with a character other than A . Then the separation between A and B in members of \mathcal{F} prevents any occurrence from matching both A and B in \mathcal{C} . In order to match \mathcal{C} at 4 positions, each occurrence must then match in a position occupied by a character from Σ_2 . Since there are only $k - 2$ such positions but $\binom{k}{2} > k - 2$ occurrences, then by the pigeonhole principle this results in a contradiction with the uniqueness of the characters in Σ_2 .

(2) Suppose \mathcal{C} contains a character from Σ_2 in position z . Then at most one occurrence matches \mathcal{C} at position z . Consider the vertex group \mathcal{V}_z . Any occurrence from \mathcal{V}_z that matches \mathcal{C} out-of-phase must determine a unique character, since it cannot match both A and B . Suppose some occurrence from \mathcal{V}_z matches \mathcal{C} in phase. Then it will not match \mathcal{C} at position z and therefore must determine a unique character. Since all occurrences from \mathcal{V}_z determine unique characters, and $|\mathcal{V}_z| = k - 1$, at most one occurrence can match \mathcal{C} at 4 positions, contradicting the fact that \mathcal{C} is a center for \mathcal{F} .

(3) Suppose some occurrence matches the center out-of-phase, then that occurrence cannot match both A and B and so must match some position containing a character from Σ_2 , contradicting Part 2.

(4) Suppose \mathcal{C} has been partially determined by occurrences from $\mathcal{V}'_z \subset \mathcal{V}_z$, for vertex position z . Consider occurrence \mathcal{I} from $S_{zx} \in \mathcal{V}_z \setminus \mathcal{V}'_z$. By Parts (2) and (3), \mathcal{I} must match the alignment positions, and positions z and x . Since \mathcal{I} is the

only member of \mathcal{V}_z that can determine a non-unique character at position x , that position has not yet been determined. In order for \mathcal{I} to match 4 positions of \mathcal{C} , \mathcal{I} must determine position x in \mathcal{C} . The first occurrence determines 4 positions in the center, and the remaining $k - 2$ occurrences each determine an additional position, a total of $k + 2$ positions. \square

Lemma 3.2 *CLIQUE parametrically reduces to CAS(m, l).*

Proof. The reduction runs in time that is fixed-parameter tractable relative to m , l , and d , so we need only show that the reduction is correct. Suppose there is a k -clique in G . Given the vertices in a clique, place their corresponding characters from Σ_1 in ascending order between characters A and B . It is easy to verify that the resulting string is a center for \mathcal{F} . Conversely, suppose there is no k -clique in G and there is a center \mathcal{C} for those strings in vertex group \mathcal{V}_z where vertex v occupies vertex position z . By Part (4) of the Lemma 3.1, occurrences from \mathcal{V}_z completely determine \mathcal{C} . Consider any set of vertices N , $|N| = k - 1$, neighboring v in G . Since there is no k -clique, some pair of vertices $a, b \in N$ are not adjacent in G . By the construction of \mathcal{F} , for any pair i, j of positions, no length- l substring of S_{ij} can have both character a at position i and b at positions j . Therefore \mathcal{C} is not a center for \mathcal{F} . \square

Theorem 3.3 *CAS(m, l) is W[1]-hard.*

Proof. Follows from Lemma 3.2 and the W[1]-hardness of CLIQUE [26]. \square

3.2.2 W[2]-hardness of CAS(l)

To show W[2]-hardness for CAS(l), we give a parameterized reduction from the W[2]-complete problem DOMINATING CLIQUE [26]. Let $G = (V, E)$ be a graph for which we wish to determine whether G has a dominating clique of size k . We construct a family \mathcal{F} of m strings, over alphabet Σ , that has a common approximate substring of length l and distance d if, and only if, G contains a dominating clique of size k . Assume for convenience that the vertex set of G is $V = \{1, \dots, x\}$. The alphabet and substring gadgets are exactly the same as for the reduction in Section 3.2.1.

Target parameters The number of strings in \mathcal{F} is $m = f_1(k, G) = \binom{k}{2} + |V|$, which is no longer independent of $|G|$. The functions f_2 to f_4 remain as defined for

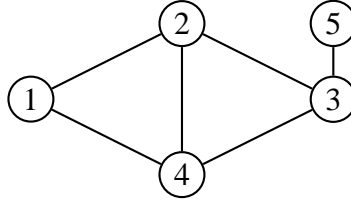


Figure 3.3: This is example graph 2. The vertex set $\{2, 3, 4\}$ forms a dominating 3-clique.

the reduction in Section 3.2.1.

The reduction The strings will form two groups $\mathcal{F} = \mathcal{F}_{G_E} \cup \mathcal{F}_{G_V}$ having distinct roles. The $\binom{k}{2}$ strings in \mathcal{F}_{G_E} are exactly those described in the previous reduction. These have the same role: determining a center that corresponds to a k -clique in G .

The strings of \mathcal{F}_{G_V} are responsible for verifying that any center determined by occurrences from \mathcal{F}_{G_E} corresponds not only to a k -clique, but to a dominating set as well:

$$\mathcal{F}_{G_V} = \{S_{V_p} : 1 \leq p \leq |V|\}.$$

String S_{V_p} is composed of all edge components having the character $q \in \Sigma_1$ such that q is a neighbor of p . The components are arranged in the following manner (where product notation refers to concatenation and for any vertex x , $N[x]$ is the set of neighbors of x):

$$S_{V_p} = \prod_{\substack{q \in N[p] \\ q' \in N[q] \\ 1 \leq i < j \leq k}} \begin{cases} \langle \text{edge}(i, j)(q', q) \rangle \langle \text{separator} \rangle & \text{if } q' < q, \\ \langle \text{edge}(i, j)(q, q') \rangle \langle \text{separator} \rangle & \text{if } q < q'. \end{cases}$$

An example of this reduction for the graph in Figure 3.2.2 and a desired dominating clique size of 3 can be seen in Figure 3.2.2.

Lemma 3.3 DOMINATING CLIQUE *parametrically reduces to CAS(l)*.

Proof. The construction described above runs in time that is fixed-parameter tractable relative to l and d , so we need only show that the reduction is correct. As was shown

$$\begin{aligned}
S_{12}: & A12uBu^5 A14uBu^5 A23uBu^5 A24uBu^5 A34uB \\
S_{13}: & A1u4Bu^5 A2u4Bu^5 A3u5B \\
S_{23}: & Au23Bu^5 Au24Bu^5 Au34Bu^5 Au35B \\
\\
S_{V_1}: & A12uBu^5 A14uBu^5 A23uBu^5 A24uBu^5 A34uB \\
& A1u4Bu^5 A2u4Bu^5 Au23Bu^5 Au24Bu^5 Au34B \\
S_{V_2}: & A12uBu^5 A14uBu^5 A23uBu^5 A24uBu^5 A34uB \\
& A1u4Bu^5 A2u4Bu^5 A3u5Bu^5 Au23Bu^5 Au24B \\
& Au34Bu^5 Au35B \\
S_{V_3}: & A12uBu^5 A14uBu^5 A23uBu^5 A24uBu^5 A34uB \\
& A1u4Bu^5 A2u4Bu^5 A3u5Bu^5 Au23Bu^5 Au24B \\
& Au34Bu^5 Au35B \\
S_{V_4}: & A12uBu^5 A14uBu^5 A23uBu^5 A24uBu^5 A34uB \\
& A1u4Bu^5 A2u4Bu^5 A3u5Bu^5 Au23Bu^5 Au24B \\
& Au34Bu^5 Au35B \\
S_{V_5}: & A23uBu^5 A34uBu^5 A3u5Bu^5 Au23Bu^5 Au34B \\
& Au35B
\end{aligned}$$

Figure 3.4: CAS(l) representation for example graph 2 (with desired dominating clique size $k = 3$).

in Lemma 3.2, a center for \mathcal{F}_{G_E} can be obtained from any k -clique in G . Suppose some $V' \subset V$ is both a k -clique and a dominating set for G . For all vertices $p \in V$, there exists vertex $q \in V'$ such that $pq \in E$. The substring of S_{V_p} that encodes any of the $k - 1$ clique edges incident on vertex p will serve as an occurrence for the center. Therefore a dominating k -clique in G implies a center for \mathcal{F} .

Conversely, the absence of a k -clique in G implies the absence of a center for \mathcal{F}_{G_E} . Suppose no k -clique is also a dominating set in G . If there is a clique in G , there will be some vertex $p \in V$ having no neighbors in the clique. For all substrings of $S_{V_p} \in \mathcal{F}_{G_V}$, none will correspond to an edge in the clique and therefore none will match the center sufficiently to be an occurrence. \square

Theorem 3.4 CAS(l) is W[2]-hard.

Proof. Follows from Lemma 3.3 and the W[2]-hardness of DOMINATING CLIQUE [26]. \square

3.2.3 W[2]-hardness of $\text{CAS}(m, |\Sigma|)$

To show $W[2]$ -hardness for $\text{CAS}(m, |\Sigma|)$, we give a parameterized reduction from the $W[2]$ -complete problem SET COVER [26]. This result both strengthens and complements the $W[1]$ -hardness result given in [32] for $\text{CAS}(m, |\Sigma|)$ when $|\Sigma| = 2$.

Let $I = \langle \mathcal{B}, \mathcal{L} \rangle$ be an instance of SET COVER. Without loss of generality, assume that the elements of \mathcal{B} are the integers $[1, |\mathcal{B}|]$. We show how to construct an instance \mathcal{F} of $\text{CAS}(m, |\Sigma|)$ such that I has a cover of size k if and only if \mathcal{F} has a center with a particular maximum distance to any occurrence.

Target parameters The number of strings in \mathcal{F} is $m = f_1(k) = 2k$. The length of the center \mathcal{C} is $l = f_2(\mathcal{L}, k) = k|\mathcal{B}| + 2$, and the maximum distance between occurrence and center is $d = f_3(\mathcal{L}, k) = (k - 1)|\mathcal{B}|$. The maximum length of the strings in \mathcal{F} is $n = f_4(\mathcal{L}, k) = 2(k|\mathcal{B}| + 2) \cdot |\mathcal{L}| - 1$, and the alphabet size is $|\Sigma| = f_5(k) = 3k + 1$.

The alphabet The string alphabet is $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \{A\}$. We refer to these as *solution characters* (Σ_1), *unique characters* (Σ_2) and the *alignment character* (A), with

$$\begin{aligned}\Sigma_1 &= \{s_1, \dots, s_k\}, \\ \Sigma_2 &= \{u_{11}, u_{12}, u_{21}, u_{22}, \dots, u_{k1}, u_{k2}\}.\end{aligned}$$

For $1 \leq i \leq k$, we assume without loss of generality that character s_i is the integer i . The characters of Σ_2 , denoted by u with subscripts, are identical within a string, but different between strings.

Substring gadgets We next describe the three “high level” component substrings used in the construction. For Membership Indicators, the product is ordered and refers to concatenation.

Fillers:

$$\langle \text{Filler}(i) \rangle = s_i^{(k-1)|\mathcal{B}|}$$

Separators:

$$\langle \text{Separator}(i, p) \rangle = u_{ip}^{(k|\mathcal{B}|+2)}$$

Subset Indicators:

$$\langle \text{Subset}(i, j, p) \rangle = \prod_{b \in \mathcal{B}} g(i, j, p, b)$$

$$\begin{aligned}
\mathcal{B}: & \{1, 2, 3, 4, 5, 6, 7\} \\
\mathcal{L}_1: & \{1, 4, 6\} \\
\mathcal{L}_2: & \{1, 2, 4\} \\
\mathcal{L}_3: & \{3, 5\} \\
\mathcal{L}_4: & \{1, 2, 3, 7\} \\
k: & 3
\end{aligned}$$

Figure 3.5: Example instance of SET COVER.

The Fillers are strings of length $(k-1)|\mathcal{B}|$ and each corresponds to some $\mathcal{L}_i \in \mathcal{L}$. The Separators are strings of length $k|\mathcal{B}| + 2$. Each is comprised entirely of characters from Σ_2 , and the variable p takes values from $\{1, 2\}$. The Subset Indicators are used to indicate the sets that make up a cover. The function g is defined as

$$g(i, j, p, b) = \begin{cases} s_i & \text{if } b \in \mathcal{L}_j, \\ u_{ip} & \text{otherwise.} \end{cases}$$

The reduction Each of the k sets in the solution R of I is represented by a pair of strings in \mathcal{F} . In particular, the occurrences in strings $S_{i1}, S_{i2} \in \mathcal{F}$ correspond to the i^{th} set in R . Define

$$S_{i1} = \prod_{1 \leq j \leq |\mathcal{L}|} A\langle \text{Subset}(i, j, 1) \rangle \langle \text{Filler}(i) \rangle A\langle \text{Separator}(i, 1) \rangle,$$

$$S_{i2} = \prod_{1 \leq j \leq |\mathcal{L}|} A\langle \text{Subset}(i, j, 2) \rangle \langle \text{Filler}(i) \rangle A\langle \text{Separator}(i, 2) \rangle.$$

The family of strings is then $\mathcal{F} = \{S_{11}, S_{12}, S_{21}, S_{22}, \dots, S_{k1}, S_{k2}\}$. Note that no matter what set of substrings is taken as occurrences of a center, aside from the positions containing alignment characters, any position has at most two strings with the same character. An example of this reduction is provided in Figures 3.2.3, 3.2.3, and 3.2.3. For Figure 3.2.3, the subscripts are left out of the unique characters; these are given unique symbols in Figure 3.2.3.

Define \bar{d} as the minimum possible value of d for a set of occurrences. The proof of correctness for this reduction rests on a function \hat{d} that provides a lower bound on \bar{d} . For a collection of potential occurrences of a center string for \mathcal{F} , define z_{ij} as the indicator function that has the value 1 if $S_i[j]$ is *not* the column majority character in

$$\begin{aligned}
S_{11} &: \underline{A1uu1u1u1}^{14}Au^{23}A11u1uuu1^{14}Au^{23}Auu1u1uu1^{14}Au^{23}A111uuu11^{14}Au^{23} \\
S_{12} &: \underline{A1uu1u1u1}^{14}Au^{23}A11u1uuu1^{14}Au^{23}Auu1u1uu1^{14}Au^{23}A111uuu11^{14}Au^{23} \\
S_{21} &: A2uu2u2u2^{14}Au^{23}A22u2uuu2^{14}Au^{23}\underline{Auu2u2uu2}^{14}Au^{23}A222uuu22^{14}Au^{23} \\
S_{22} &: A2uu2u2u2^{14}Au^{23}A22u2uuu2^{14}Au^{23}\underline{Auu2u2uu2}^{14}Au^{23}A222uuu22^{14}Au^{23} \\
S_{31} &: A3uu3u3u3^{14}Au^{23}A33u3uuu3^{14}Au^{23}Auu3u3uu3^{14}Au^{23}\underline{A333uuu33}^{14}Au^{23} \\
S_{32} &: A3uu3u3u3^{14}Au^{23}A33u3uuu3^{14}Au^{23}Auu3u3uu3^{14}Au^{23}\underline{A333uuu33}^{14}Au^{23}
\end{aligned}$$

Figure 3.6: $CAS(m, |\Sigma|)$ representation for the example instance of SET COVER given in Figure 5. The character u denotes a character that differs across strings. The underlined substrings are expanded and explained in Figure 3.2.3.

column j of the aligned occurrences and the value 0 otherwise (in case there are two or more characters that may serve as column majority, one is arbitrarily selected). Given z_{ij} , function \hat{d} is defined as follows:

$$\hat{d} = \left\lceil \frac{\sum_{j=1}^l \sum_{i=1}^m z_{ij}}{m} \right\rceil.$$

Lemma 3.4 $\hat{d} \leq \bar{d}$.

Proof. Let \mathcal{F} be a set of strings and X be a set of occurrences for an optimal center \mathcal{C} for \mathcal{F} . The character at each position j will cause at least $m - \sum_{i=1}^m z_{ij}$ mismatches between \mathcal{C} and members of X . Since there are l positions in \mathcal{C} , there are at least $\sum_{i=1}^m \sum_{j=1}^l z_{ij}$ mismatches between \mathcal{C} and members of X . The largest distance between \mathcal{C} and any member of X must be at least the smallest integer not less than the arithmetic mean of the total distance, which is $\lceil (\sum_{i=1}^m \sum_{j=1}^l z_{ij}) / m \rceil$. \square

As a corollary to Lemma 3.4, the following can be demonstrated by substituting the appropriate values into the formula for \hat{d} .

Lemma 3.5 *If the column majority character occurs at most twice in any column, then $\bar{d} \geq l - \frac{2l}{m}$.*

Lemma 3.6 *Let \mathcal{F} be a set of strings constructed as described in the reduction and let \mathcal{C} be a center for \mathcal{F} . Then \mathcal{C} must begin and end with the alignment character, and so must all occurrences.*

```

S11[1]:      A1aa1a1a1111111111111111A
S12[1]:      A1bb1b1b1111111111111111A
S21[93]:     Acc2c2cc222222222222222A
S22[93]:     Add2d2dd222222222222222A
S31[139]:    A333eee33333333333333333A
S32[139]:    A333fff33333333333333333A

Center String:  A333121311111222222333A

```

$$d = (k - 1)|\mathcal{B}| = 14.$$

Figure 3.7: Expanded diagrams of the substrings underlined in Figure 3.2.3, along with an optimal center string for the instances. Underlined characters are those matching corresponding positions in the center.

Proof. Suppose the center \mathcal{C} does not begin with the alignment character; then by the separation between alignment characters in members of \mathcal{F} , no occurrence can match two alignment characters in \mathcal{C} . As all but one column has at most two occurrences of the column majority character, we can rewrite the bound on \bar{d} given in Lemma 3.5 with the substitutions $m = 2k$ and $l = k|\mathcal{B}| + 2$ to obtain $\bar{d} \geq (k - 1)|\mathcal{B}| + \frac{k-1}{k}$. A symmetric argument establishes that \mathcal{C} must end with the alignment character.

Suppose some occurrence begins or ends with a character other than the alignment character. Then that occurrence cannot match \mathcal{C} at those positions. Again using Lemma 3.5, $\bar{d} \geq (k - 1)|\mathcal{B}| + \frac{k-1}{k}$. \square

Lemma 3.7 SET COVER *parametrically reduces to* CAS($m, |\Sigma|$).

Proof. The construction described above runs in time that is fixed-parameter tractable relative to m and $|\Sigma|$. Hence, we need only show that the reduction is correct.

Suppose there is a cover R for \mathcal{B} , such that $|R| = k$. From R , construct a center \mathcal{C} for \mathcal{F} as follows. (1) The first and last positions of \mathcal{C} are assigned the alignment character A. (2) The next $|\mathcal{B}|$ positions, used to represent elements of the base set, are each assigned a character indicating one of the sets in R that covers the corresponding element. For each $b \in \mathcal{B}$, choose some $\mathcal{L}_i \in R$, such that $b \in \mathcal{L}_i$,

as covering b . Since R is a cover for \mathcal{B} , there is at least one such choice for every $b \in \mathcal{B}$. If \mathcal{L}_i is chosen to cover b , then s_i is assigned to position $b+1$ in \mathcal{C} (recall that the elements of \mathcal{B} have been equated with the integers 1 to $|\mathcal{B}|$). (3) The remaining $(k-1)|\mathcal{B}|$ positions of \mathcal{C} correspond to the Filler gadgets. For each $\mathcal{L}_i \in R$, if x_i positions ($0 \leq x_i \leq |\mathcal{B}|$) of \mathcal{C} have been assigned characters corresponding to elements in \mathcal{L}_i , then $|\mathcal{B}| - x_i$ positions in the Filler part of \mathcal{C} are assigned the character s_i . If $\mathcal{L}_j \in \mathcal{L}$ is the i^{th} set in R , then the substring of S_{i1} (and of S_{i2}) that begins and ends with the alignment character, and contains the j^{th} Subset Indicator, matches \mathcal{C} in exactly $2 + |\mathcal{B}|$ positions. Therefore \mathcal{C} is a center for \mathcal{F} with distance exactly $(k-1)|\mathcal{B}|$ to any occurrence.

Suppose there is a $(k-1)|\mathcal{B}|$ center \mathcal{C} for \mathcal{F} constructed as per the reduction from an instance of SET COVER. By Lemma 3.5, the column majority character occurs at least twice in each column, implying all but the first and last positions of \mathcal{C} are occupied by solution characters (*i.e.* from Σ_1). For each distinct i and i' , the occurrences of \mathcal{C} from S_{i1} and $S_{i'1}$ match \mathcal{C} at distinct sets of positions. Consider the k occurrences of \mathcal{C} from S_{i1} , $1 \leq i \leq k$. The sets of positions in \mathcal{C} that are matched by these occurrences must collectively match all positions of \mathcal{C} . Hence the corresponding subsets from \mathcal{L} cover all elements of \mathcal{B} . \square

Theorem 3.5 $\text{CAS}(m, |\Sigma|)$ is $W[2]$ -hard.

Proof. Follows from Lemma 3.7 and the $W[2]$ -hardness of SET COVER [26]. \square

3.2.4 Other hardness results

The next result, due to Todd Wareham [91], indicates that unless $P = NP$, there is no algorithm for COMMON APPROXIMATE SUBSTRING with time complexity $O((mn)^{f(|\Sigma|)})$ or $O((mn)^{f(l-d)})$.

Theorem 3.6 $\text{CAS}(|\Sigma|)$ and $\text{CAS}(l-d)$ are not in XP unless $P = NP$.

Proof. Follows from the NP-hardness of CLOSEST STRING when $|\Sigma| = 2$ [34], and the NP-hardness of COMMON APPROXIMATE SUBSTRING when $l-d = 4$ demonstrated by the reduction from CLIQUE in Section 3.2.1. \square

3.3 Membership in the W-Hierarchy

To show inclusion of parameterized variants of COMMON APPROXIMATE SUBSTRING in classes of the W-Hierarchy, solution checking circuits of limited *weft* are required. The following three different circuits use distinct strategies to test solutions for various parameterizations of CAS. These are called the *center testing circuit*, the *instance testing circuit* and the *single occurrence + modifications testing circuit*. Diagrams for these circuits are provided in Appendix A.

Center testing circuit Define $\mathcal{F} = \{S_1, \dots, S_m\}$, let $\langle \mathcal{F}, l, d \rangle$ be an instance of CAS, and let \mathcal{C} be any center for \mathcal{F} . The j^{th} length- l substring of S_i is denoted $S_i[j]$, and position p of $S_i[j]$ is denoted $S_i[j][p]$. The set X is used to index size $l - d$ subsequences of a length- l string. Note that each X_p is an *ordered* $(l - d)$ -tuple:

$$X = \left\{ X_p : X_p \subset (1, \dots, l), |X_p| = l - d, 1 \leq p \leq \binom{l}{d} \right\}.$$

Let the set of variables

$$A = \left\{ a[i, j, p, q] : 1 \leq i \leq m, 1 \leq j \leq n - l + 1, 1 \leq p \leq |X|, 1 \leq q \leq l - d \right\}$$

denote position q in $X_p \cap S_i[j]$. Let $B = \{b[u, v] : 1 \leq u \leq l, 1 \leq v \leq |\Sigma|\}$ be a set of Boolean variables. The intended interpretation of variable $b[u, v]$ is that character v occupies position u in \mathcal{C} . The variable $a[i, j, p, q]$ takes the value of $b[u, v]$ if and only if position q of X_p is u and that position is occupied by character v in $S_i[j]$. Otherwise $a[i, j, p, q]$ is set to *false*.

Let $E = E_1 E_2$ be the Boolean expression over the set of variables B , where:

$$E_1 = \prod_{u=1}^l \prod_{1 \leq v < v' \leq |\Sigma|} (\neg b[u, v] + \neg b[u, v']),$$

$$E_2 = \prod_{i=1}^m \sum_{j=1}^{n-l+1} \sum_{p=1}^{|X|} \prod_{q=1}^{l-d} a[i, j, p, q].$$

For example consider the set of strings $S_1 = \text{tgg}tca$, $S_2 = \text{acc}gac$, and $S_3 = \text{cg}gtag$ over alphabet $\Sigma = \{a, c, g, t\}$. We assume the order $a = 1$, $c = 2$, $g = 3$ and $t = 4$ on Σ . If $X_p = (1, 2, 3)$, then $a[1, 1, p, 1] = b[1, 4]$ because both correspond to the

character t at position 1 in a length- l string. Similarly, $a[3, 1, p, 1] = b[1, 2]$, corresponding to c at position 1. If $p = (1, 3, 5)$, $a[2, 2, p, 2] = b[4, 3]$, corresponding to g at position 4.

The purpose of E_1 is to force a correspondence between satisfying interpretations and strings over Σ^l . Notice that a weight l interpretation falsifies E_1 if more than one $b[i, j]$ is assigned true for any i .

We claim that E has a weight l truth assignment if, and only if, there exists a center \mathcal{C} for \mathcal{F} . If \mathcal{C} exists, then the assignment

$$\mathcal{C}[p] = S_i[j][p] = q \Leftrightarrow a[i, j, p, q] = \text{true}$$

satisfies E . Conversely, let \mathcal{T} be a weight l satisfying truth assignment for E . The clauses of E_1 ensure that \mathcal{T} indicates a unique string $\mathcal{C} \in \Sigma^l$. The clauses of E_2 ensure that for each i , some substring $S_i[j]$ matches $l - d$ positions of \mathcal{C} . This implies that in each S_i , there is a substring of length l that is distance less than d from \mathcal{C} . Therefore \mathcal{C} is a center for \mathcal{F} .

Theorem 3.7 $\text{CAS}(l) \in W[2]$ and $\text{CAS}(m, l) \in W[1]$.

Proof. Follows by observing that when l is fixed, the center testing circuit has weft 2. If m is fixed along with l , the center testing circuit has weft 1. \square

Occurrence testing circuit We construct a new circuit, called the *occurrence testing circuit*, having little resemblance to the center testing circuit. Our goal here is to show membership for versions of CAS when l is left free. The idea is to select m occurrences and, for each occurrence, d positions where the occurrence is exempted from having to match a center. The circuit is only a slight modification of a circuit that solves the length- l common substring problem.

Let $B = \{b[i, j] : 1 \leq i \leq m, 1 \leq j \leq n - l + 1\}$ be a set of Boolean variables with the intended interpretation that $b[i, j]$ is set true when $S_i[j]$ is an occurrence of \mathcal{C} . Let $W = \{w[i, r, p] : 1 \leq i \leq m, 1 \leq r \leq d, 1 \leq p \leq l\}$ be a set of Boolean variables with the intended meaning that any occurrence of \mathcal{C} in S_i need not match \mathcal{C} at position p . The index r is used to restrict the number of such *exemptions* to be not more than d for any occurrence. In the description of the circuit, the set of variables $A = \{a[i, j, p, q] : 1 \leq i \leq m, 1 \leq j \leq n - l + 1, 1 \leq p \leq l, 1 \leq q \leq |\Sigma|\}$ acts as an alias for the variables from B . As in the description of the *center testing circuit*, for any appearance of the variable $a[i, j, p, q]$, the substitution $a[i, j, p, q] \leftarrow b[i, j]$ is assumed exactly when $S_i[j]$ has character q at position p . Otherwise $a[i, j, p, q]$ takes value *false*.

Let $E = E_1 E_2 E_3$ be the Boolean expression over the set of variables of $B \cup W$, where:

$$\begin{aligned}
 E_1 &= \prod_{i=1}^m \prod_{1 \leq j < j' \leq n-l+1} (\neg b[i, j] + \neg b[i, j']), \\
 E_2 &= \prod_{i=1}^m \prod_{r=1}^d \prod_{1 \leq p < p' \leq l} (\neg w[i, r, p] + \neg w[i, r, p']), \\
 E_3 &= \prod_{p=1}^l \sum_{q=1}^{|\Sigma|} \prod_{i=1}^m \left(\sum_{r=1}^d w[i, r, p] + \sum_{j=1}^{n-l+1} a[i, j, p, q] \right).
 \end{aligned}$$

We claim that E has a weight $m + md$ satisfying truth assignment if, and only if, there is a center \mathcal{C} for \mathcal{F} . Given center \mathcal{C} , a satisfying truth assignment for E can be obtained by setting $b[i, j]$ to true for each occurrence $S_i[j]$ of \mathcal{C} , and also setting $w[i, r, p]$ to true if the r^{th} mismatch in the occurrence from S_i occurs at position p . For the converse case, let \mathcal{T} be a weight $m + md$ satisfying truth assignment for E . The clauses of E_1 ensure that \mathcal{T} corresponds to at most m occurrences, one from each S_i . The clauses of E_2 ensure that at most d mismatching positions are selected for the occurrence from any S_i . E_1 and E_2 combined force \mathcal{T} to correspond directly to a set of occurrences and, for each occurrence, a set of positions where the occurrence may differ from a center. The fact that \mathcal{T} satisfies E_3 implies that all occurrences agree in all positions with the possible (and permitted) exception of the exempted positions. Hence \mathcal{F} has a center.

Theorem 3.8 $\text{CAS}(m, d) \in W[3]$ and $\text{CAS}(m, |\Sigma|, d) \in W[2]$.

Proof. Follows by observing that when m and d are fixed, the occurrence testing circuit has weft 3. If, additionally, $|\Sigma|$ is fixed, the weft of the occurrence testing circuit is reduced by one. \square

Single occurrence + modifications testing circuit The idea behind this circuit comes from the observation that a center can be obtained by isolating an arbitrary string from \mathcal{F} (we use S_1), and applying substitutions for characters in up to d positions in each substring $S_1[j]$ of S_1 . We use a guess and test strategy: first guess a center by selecting some $S_1[j]$, along with the positions and characters by which the center differs from $S_1[j]$, then test the center by searching for an occurrence in each S_i , $2 \leq i \leq m$. The goal here is to have a weight $d + 1$ truth assignment represent the selection of some j ($1 \leq j \leq n - l + 1$), and d substitutions to positions of $S_1[j]$ that transform $S_1[j]$ into a center.

To describe the input to the circuit, we use the following sets of variables:

$$\begin{aligned} X_1 &= \{x_1[i, j, p, r] : 1 \leq i \leq m, 1 \leq j \leq n-l+1, 1 \leq p \leq l, 1 \leq r \leq |\Sigma|\}, \\ X_2 &= \{x_2[j] : 1 \leq j \leq n-l+1\}, \\ X_3 &= \{x_3[p, r] : 1 \leq p \leq l, 1 \leq r \leq |\Sigma|\}. \end{aligned}$$

where the value of $x_1[i, j, p, r]$ corresponds to the truth of $S_i[j]$ being occupied by character r at position p (these values are fixed for each occurrence and are not part of a truth assignment). The variables of X_1 are actually constant for each instance of CAS, and do not affect the weight of the input to the circuit. The weight $d+1$ truth assignment comes from selecting exactly one member of X_2 (representing a *single occurrence*: a substring of S_1) and d members of X_3 (representing the *modifications*). Once the center has been “guessed”, it remains to test it against potential occurrences from the other strings in \mathcal{F} . Unlike the *center testing circuit* above, l is not fixed, so a different strategy is necessary to test the “guessed” center.

The set of variables $\{g[p, r] : 1 \leq p \leq l, 1 \leq r \leq |\Sigma|\}$ describes the “guessed” center, where

$$g[p, r] = \left(\sum_{j=1}^{n-l+1} x_2[j] \cdot x_1[1, j, p, r] \right) \cdot \left(\prod_{\substack{1 \leq r' \leq |\Sigma| \\ r' \neq r}} \neg x_3[p, r'] \right) + x_3[p, r].$$

The lower layers of the circuit are described by the variables

$$B = \left\{ b[i, j, p] : 2 \leq i \leq m, 1 \leq j \leq n-l+1, 1 \leq p \leq l \right\},$$

with the interpretation that $b[i, j, p] = \text{true}$ if, and only if, $S_i[j]$ matches the guessed center at position p or is one of at most d mismatches.

Members of B occur at different depths. We stack the variables of B so that $b[i, j, p]$ may depend on the value of $b[i, j, p-1]$. The purpose of this is to prevent having to count the number of mismatches (between the guessed center and an occurrence) at a single level. To do so would introduce an exponential number of gates. The strategy we use is to maintain a count of the amount of permitted mismatches, a count that is decremented each time a mismatch occurs. The set of variables A implement counters, one for each $S_i[j], i \neq 1$:

$$A = \left\{ a[i, j, p, q] : 2 \leq i \leq m, 1 \leq j \leq n-l+1, 0 \leq p \leq l, 1 \leq q \leq d+1 \right\},$$

such that

$$a[i, j, p, q] = \left(a[i, j, p-1, q] \cdot \sum_{r=1}^{|\Sigma|} (g[p, r] \cdot x_1[i, j, p, r]) \right) + a[i, j, p-1, q+1].$$

For all i, j and p , the value of $a[i, j, p, d+1]$ is set to *false*, and for all i, j and q , the value of $a[i, j, 0, q]$ is set to true.

We now define the variables of B :

$$b[i, j, p] = a[i, j, p, 1] + \sum_{r=1}^{|\Sigma|} (g[p, r] \cdot x_1[i, j, p, r]).$$

With the variables of B already determined, the circuit C is described by expression $E = E_1 E_2 E_3$, defined as:

$$\begin{aligned} E_1 &= \prod_{1 \leq j < j' \leq n-l+1} (\neg x_2[j] + \neg x_2[j']), \\ E_2 &= \prod_{p=1}^l \prod_{1 \leq r < r' \leq |\Sigma|} (\neg x_3[p, r] + \neg x_3[p, r']), \\ E_3 &= \prod_{i=2}^m \sum_{j=1}^{n-l+1} \prod_{p=1}^l b[i, j, p]. \end{aligned}$$

The circuit is satisfied if, and only if, some “guess” matches at least $l-d$ positions in at least one substring for every member of \mathcal{F} . The size of the circuit is $\Theta(nml(|\Sigma| + d))$. The depth of the circuit is $\Theta(l)$ since, for each i, j and r , there is a path passing through $a[i, j, 1, r], a[i, j, 2, r] \dots a[i, j, l, r]$.

Theorem 3.9 $CAS(d) \in W[P]$.

Proof. Follows from the fact that when d is fixed and all other parameters left free, the *single occurrence + modifications circuit* has weft $\Theta(l)$. \square

3.4 Summary and Open Problems

In this chapter, the pattern finding problem was abstracted as the parameterized problem COMMON APPROXIMATE SUBSTRING, and analyzed according to parameterized complexity theory. The complexities of many parameterized variants of

Table 3.1: Parameterized complexity map for COMMON APPROXIMATE SUBSTRING. Major results originating in this thesis are underlined. These imply nearly all other results in the table, most of which were not previously known. The W[1]-hardness of $CAS(m, l)$ was discovered independently in [32].

<i>Parameter</i>	–	Σ	m	m, Σ
–	NP-complete	\notin XP	W[2]-hard	<u>W[2]-hard</u>
d	W[2]-hard, <u>W[P]</u>	W[P]	W[1]-hard, <u>W[3]</u>	<u>W[2]</u>
l	<u>W[2]-complete</u>	FPT	<u>W[1]-complete</u>	FPT
n	<u>FPT</u>	FPT	FPT	FPT

the problem have been resolved. These results, along with all results implied by the dependencies $d \leq l \leq n$, are summarized in Table 3.1. With the exception of the $CAS(|\Sigma|)$ and $CAS(|\Sigma|, l)$ results, all results depicted in Table 3.1 originated in this thesis. The W[1]-hardness result for $CAS(l, m)$ was obtained independently in [32].

The map depicted in Table 3.1 is nearly complete with respect to indicating parameterized tractability or hardness. There remain two cells ($CAS(m, |\Sigma|, d)$ and $CAS(|\Sigma|, d)$) for which no hardness or FPT result has been obtained. Along with these two *completely open* problems, there are many variants for which the derived bounds may be tightened. Those variants found to be in FPT, not in XP, or complete for some class W[t] are as tight as possible under the theory of parameterized complexity. The following problems present interesting prospects for future research.

- No difference between the complexity of having fixed d and the complexity of fixed l has been demonstrated. Intuitively, such a difference should exist. All variants with l fixed are “complete”, and the only way to show a difference between the effects of l and d is to raise the lower bounds on the complexity of the variants parameterized with d and not l .
- For those cases when d is not fixed (the first row of Table 3.1), no upper bounds have been obtained. Is it possible that these problems *cannot* be expressed as a WEIGHTED CIRCUIT SATISFIABILITY problem? As a general remark on the theory of parameterized complexity, the difficulty of characterizing certain problem variants using parameterized circuits may indicate an inherent limitation of the W hierarchy as a model for non-deterministic parameterized computation.
- $CAS(m, d)$ has been shown to be W[1]-hard and in W[3]. This is an unusual

result and further effort might show it complete for $W[1]$ or $W[2]$. If it is shown complete for $W[3]$, it will be the first such problem.

- Is it possible that $CAS(m, |\Sigma|, d)$ resides in FPT? What about $CAS(|\Sigma|, d)$?

The last of these is certainly the most important to solving these problems in practice. It appears as though resolving the complexity of $CAS(m, |\Sigma|, d)$ or $CAS(|\Sigma|, d)$ will require a key structural fact about the instances of CAS; one that is not presently known.

Chapter 4

Optimizing Aspects of Patterns

In Chapter 3 the central pattern discovery problem was analyzed according to parameterized complexity under the name COMMON APPROXIMATE SUBSTRING. With the exception of two variants, the parameterized complexity is resolved. Unfortunately, few prospects for efficient algorithms have been identified; most parameterized results indicate intractability. In this chapter, the focus remains theoretical but the tool of analysis turns to approximation theory.

In the biological context, there are cases where approximate solutions to the pattern discovery problem are useful. When distance between patterns and pattern occurrence is small, any algorithm with a small performance ratio will produce a solution with a small absolute difference with respect to the optimal solution. Also, approximation is an established tool for solving NP-hard problems, and attempts have already been made to apply it to the focal problem of this thesis [9, 54, 59]; without rigorous theoretical analysis, the quality of those attempts cannot be judged. A more important reason has to do with the information provided by the analysis. Properties of problems that are essential to obtaining performance ratios can reveal heuristics that are useful in practice. These heuristics may be used to reduce the average running time of programs, *e.g.* to eliminate large parts of the search space prior to the application of an exact algorithm.

The pattern discovery problem has been studied as an optimization problem with the objective of minimizing the value of d . The problem has been studied under the name CLOSEST SUBSTRING in [9, 54, 59, 61].

CLOSEST SUBSTRING

Instance: A set $\mathcal{F} = \{S_1, \dots, S_m\}$ of strings over alphabet Σ such that $|S_i| = n$, integer l , ($1 \leq l \leq n$).

Objective: Minimize $d = \max_{1 \leq i \leq m} \min_{1 \leq j \leq n-l+1} d_H(\mathcal{C}, s_i)$, where $|\mathcal{C}| = l$ and s_i is a substring of S_i , ($1 \leq i \leq m$).

We let $d_{\text{opt}} = \text{radius}(\mathcal{F})$, and call a string $\mathcal{C} \in \Sigma^l$ an optimal center if, and only if, $\text{radius}_{\mathcal{F}}(\mathcal{C}) = d_{\text{opt}}$.

This chapter is organized into two main parts. The first part, Section 4.1, examines issues related to the influence of alphabet size on the approximability of CLOSEST SUBSTRING. Existing positive complexity results for CLOSEST SUBSTRING depend on the assumption that the input alphabet is of constant size; we seek to determine the extent to which this assumption is necessary. Next, in Section 4.2, we look at alternative formulations of the CLOSEST SUBSTRING optimization problem. These different formulations are distinguished by their objective functions; all other objectives are shown NP-hard to approximate with performance better than a particular threshold for large alphabets. Algorithms are also given for these alternative formulations, and some tight bounds on approximability are obtained.

4.1 Closest Substring and Alphabet Size

The first mention of a provable performance ratio for CLOSEST SUBSTRING was the trivial 2-approximation in [54], which simply tests each substring of the input strings as a center and keeps the one with the best score. Proof of correctness of this approximation is given by Proposition 2.1. By itself this method is of little use, but it has formed the basis of more sophisticated methods, for instance the SP-Star method of Pevzner and Sze [71].

A PTAS for the related CLOSEST STRING problem (*i.e.* the problem obtained by setting $l = n$) was given in [57]. The algorithm has two parts, combining the technique of randomized rounding [74] with an enumerative approach. The essence of this algorithm lies in a central combinatorial result about instances of CLOSEST STRING (formally stated as Lemma 4.1 below). A PTAS was given in [61] (and described in more detail in [59]) that makes use of the above algorithm for CLOSEST STRING. The adaptation to CLOSEST SUBSTRING requires intricate probabilistic analysis. Details of these algorithms are given in Section 4.1.2.

The approximation scheme given in [61] for CLOSEST SUBSTRING was shown to have a time complexity with an exponent of

$$\Theta((1/\epsilon)^4 \ln |\Sigma|),$$

where $\epsilon > 0$ is the performance ratio of the algorithm. This positive complexity result leaves open the question of how the alphabet size contributes to the complexity of CLOSEST SUBSTRING. In many practical situations, such as the most common

in molecular biology, the size of the alphabet is known in advance. In such circumstances, the algorithms of [59] run in polynomial time, since $|\Sigma|$ is treated as a constant. In theory it is possible for the input alphabet to be a function of the input size; such large sizes are useful in mathematical constructions such as transformations. Large alphabets also allow flexibility and permit more expressive alphabets to be introduced. Because of the exponential dependence on the alphabet size, and the fact that problem instances can be constructed with large alphabets, the algorithm of [59] is not a PTAS for the general CLOSEST STRING problem.

In this section, we analyze the influence of alphabet size on the CLOSEST SUBSTRING problem as defined above. While the major result here (presented in Section 4.1.3) is a PTAS for CLOSEST SUBSTRING when no assumptions are made about the alphabet, other results are presented throughout this chapter indicating that the alphabet size does significantly influence many pattern finding problems.

From the opposite perspective, we investigate how the restriction to a specific alphabet can be exploited to develop more efficient algorithms. In Section 4.1.4 it is shown that a performance ratio of

$$1 + 2^{(1-r)} + \epsilon$$

can be obtained for CLOSEST STRING when the strings given as input are *binary*. The only previous research focusing on the problem under binary alphabet are from [37], where no properties intrinsic to binary strings are exploited, giving a $4/3 + \epsilon$ randomized approximation (subsequently improved by the results mentioned above). In Section 4.1.5 we take preliminary steps toward extending the analysis to other constant sized alphabets.

4.1.1 Additional terminology

The CLOSEST STRING problem is a simpler version of CLOSEST SUBSTRING, where $l = n$, and is solved as a subproblem by the CLOSEST SUBSTRING algorithm of [59].

CLOSEST STRING

Instance: A set $\mathcal{F} = \{S_1, \dots, S_m\}$ of strings over alphabet Σ such that $|S_i| = n$.

Objective: Minimize $d = d_H(\mathcal{C}, S_i)$, where $\mathcal{C} \in \Sigma^n$, for all $S_i \in \mathcal{F}$.

For the following few sections, we require additional terminology for discussing properties of string sets. Given a string S and an ordered set of positions $X =$

$(x_1, \dots, x_k) \subseteq (1, \dots, n)$, we define *the restriction of S to X* as the string $S|_X = S[x_1]S[x_2] \dots S[x_k]$; the string obtained by removing all positions not in X .

Given a set of strings \mathcal{F} with optimal center \mathcal{C} and some subset $R \subseteq \mathcal{F}$, we define certain (sub)sets of positions with respect to R and \mathcal{C} . When we refer to these subsets we index them with a subset R , but the particular optimal center \mathcal{C} is always implicit from context. The set of positions where some member of R does not match \mathcal{C} is called the set of *relevant positions*:

$$A_R = \{i : (1 \leq i \leq n) \wedge \exists s \in R, s[i] \neq \mathcal{C}[i]\}.$$

Inside the *relevant positions*, there are the *good positions*, denoted G_R , at which some member of R contains the same character as \mathcal{C} :

$$G_R = \{i \in A_R : \exists s \in R, s[i] = \mathcal{C}[i]\}.$$

Since $G_R \subseteq A_R$, not all members of R will match \mathcal{C} at positions in G_R . Also inside the relevant positions are the *bad positions*,

$$B_R = \{i \in A_R : \forall s \in R, s[i] \neq \mathcal{C}[i]\},$$

which are exactly those positions that are *relevant*, but not *good*. The bad positions are of two types. Those bad positions at which all members of R agree are called the *identical positions*,

$$I_R = \{i \in B_R : \forall s, s' \in R, s[i] = s'[i]\},$$

and those bad positions that are not identical in R are the *differing positions*,

$$D_R = \{i \in B_R : \exists s, s' \in R, s[i] \neq s'[i]\}.$$

Recall that if position i is identical in \mathcal{F} , then $\mathcal{C}[i] = S[i]$ (see Proposition 2.2), for all $S \in \mathcal{F}$. This allows us to ignore positions that are identical in \mathcal{F} ; we simply assume none exist. In many cases, our analysis proceeds correctly under the simplifying assumption that we may permute the positions of strings in \mathcal{F} as we wish (see Lemma 2.2). An important observation is that given only R , it is impossible to distinguish positions of G_R from positions of D_R . It is also impossible to distinguish positions of I_R from those positions that are not in A_R , unless strings in $\mathcal{F} \setminus R$ are consulted.

For any $R \subseteq \mathcal{F}$ we define the sets

$$\begin{aligned}\Sigma_R &= \{\Sigma_{R(p)} : (1 \leq p \leq n)\}, \\ \Sigma_{R(p)} &= \{\alpha \in \Sigma : \exists S \in R, s[p] = \alpha\}.\end{aligned}$$

These are *position specific* alphabets, defined for each position by the characters occupying that position in members of R . Notice that for each position $p \in G_R \cup D_R$, $|\Sigma_{R(p)}| \leq r$ regardless of $|\Sigma|$. For any $1 \leq p \leq n$ and any $\alpha \in \Sigma_{R(p)}$, there exists some $S \in R$ such that $S[p] = \alpha$. We use the symbol Σ_R^n to denote the set of strings $\{S \in \Sigma^n : \forall 1 \leq p \leq n, S[p] \in \Sigma_{R(p)}\}$.

4.1.2 Assuming constant sized alphabet

The most important theoretical result concerning the approximability of CLOSEST SUBSTRING is the PTAS of Bin Ma [61], a large part of which is based on the work of Li, Ma and Wang [58]. Our analysis suggest modifications that improve performance of the algorithm in certain situations. We only provide a high level description of their algorithm, and refer the reader to [59] for the detailed algorithm and full proofs of their results.

The PTAS for CLOSEST SUBSTRING given in [61] relied heavily on the PTAS for CLOSEST STRING as a subroutine. For this reason we begin by describing the PTAS for CLOSEST STRING. The PTAS consists of two stages. The first stage involves sampling subsets of the input strings. It is shown (by the Lemma restated below) that for each $\epsilon > 0$, there exists a size ϵ^{-1} subset of the input strings, such that the positions identical in members of that subset are likely to match an optimal center at those same positions. The analysis of the PTAS rests on this lemma, which we state in the terminology of Section 4.1.1.

Lemma 4.1 (Lemma 2.1 [59]) *For any set \mathcal{F} of strings and any $0 \leq \epsilon \leq 1$, there exists an optimal center \mathcal{C} for \mathcal{F} , and a set $R \subseteq \mathcal{F}$, such that $|R| \leq \lceil \epsilon^{-1} \rceil$ and for any $S \in \mathcal{F}$,*

$$|I_R \cap \{i : \mathcal{C}[i] \neq S[i]\}| \leq d_{\text{opt}}/(2\epsilon^{-1} - 1).$$

Hence, using information from some subset of the input strings, the problem is simplified by assuming the positions identical within R match corresponding positions of an optimal center. By Lemma 4.1, this simplification is at a small cost in terms of the objective function. With no efficient way to identify the subset described by Lemma 4.1, the algorithm must exhaustively sample all size ϵ^{-1} subsets

of \mathcal{F} , in order to find the subset with the property indicated in Lemma 4.1. When $|D_R| \in O(\ln m)$, enumeration of all possible strings in Σ_R^n obtains the approximate center implied by Lemma 4.1. When $d_{\text{opt}} > 3\epsilon^{-2} \ln m$, randomized rounding with derandomization provides the $(1 + \epsilon)$ approximation in polynomial time¹.

The PTAS for CLOSEST SUBSTRING incorporates an idea used in randomized fingerprinting methods (see *e.g.* [63]). The important idea is essentially the following, which appeared originally as part of Theorem 1 in [61].

Lemma 4.2 (loosely adapted from [61]) *If $d_{\text{opt}} > 6\epsilon^{-2} \ln(mn)$, then with high probability there is a subsequence of length $\Theta(\epsilon^{-2} \ln(mn))$ of an optimal center that can be used to identify a set of occurrences of a near optimal center.*

The importance of the above lemma is that it provides a method of obtaining the occurrences of a pattern, or a good substitute for those occurrences, without knowledge of the pattern itself. The technique is based on random sampling and using the sample mean to estimate the population mean.

Making use of the crucial results underlying the PTAS for CLOSEST STRING from [58] and CLOSEST SUBSTRING from [61] requires enumerating all strings of length $c \ln(m)$, over Σ , for some absolute constant c . This automatically introduces a factor of $\ln(|\Sigma|)$ into the exponent of the time complexity.

4.1.3 Eliminating alphabet size dependence

Here we show that CLOSEST SUBSTRING admits a PTAS even when the alphabet size is allowed to grow with the size of the input. In particular, we show that an exponent of

$$\Theta(\epsilon^{-2} \ln \epsilon^{-1})$$

in the time complexity for a CLOSEST SUBSTRING algorithm can be achieved while maintaining the performance ratio of $(1 + \epsilon)$, for any $\epsilon > 0$. While this does not directly provide a more practical solution for CLOSEST SUBSTRING, it establishes the membership of the general problem inside the class PTAS. The algorithm given in this section is a modified version of the algorithm of [61]. The proofs employ the same techniques as those found in [61] and [59]. While the result of this section can be proved with two lemmas, and an appeal to the algorithms of [61] and [59], the full algorithm is described for completeness.

¹To derandomize the algorithm, the method of conditional probabilities was cited in [58], but modified (at large cost in time complexity, even for a small alphabet) in [59].

Closest String over large alphabets

The source of the alphabet size dependence in the PTAS for CLOSEST STRING is Lemma 4.1, which does not provide any information about an optimal center inside the positions of A_R . This is a consequence of the fact that we have no knowledge of the influence of D_R ; the next lemma provides such knowledge. For any set of strings \mathcal{F} with optimal center \mathcal{C} , and any $R \subseteq \mathcal{F}$, define the function

$$\varphi(R, \mathcal{C}) = d_{\text{opt}} - \min_{\mathcal{C}' \in \Sigma_R^n} (d_H(\mathcal{C}, \mathcal{C}')).$$

In what follows we make implicit use of the observation that for all $R, R' \subseteq \mathcal{F}$,

$$\varphi(R \cup R', \mathcal{C}) \leq \varphi(R, \mathcal{C}) + \varphi(R', \mathcal{C}).$$

Lemma 4.3 *Let \mathcal{F} be a set of strings, and $R \subseteq \mathcal{F}$. If for every $\mathcal{C}' \in \Sigma_R^n$, there exists some $S \in \mathcal{F}$ such that $d_H(S, \mathcal{C}') > \rho d_{\text{opt}}$, then for any optimal center \mathcal{C} for \mathcal{F} , there exists a string $S \in \mathcal{F}$ such that $\varphi(R \cup \{S\}, \mathcal{C}) - \varphi(R, \mathcal{C}) > (\rho - 1)d_{\text{opt}}$.*

Proof. We prove the contrapositive. Suppose there exists a center \mathcal{C} for \mathcal{F} such that for every $S \in \mathcal{F}$, $\varphi(R \cup \{S\}, \mathcal{C}) - \varphi(R, \mathcal{C}) < (\rho - 1)d_{\text{opt}}$. Select an arbitrary string $S \in \mathcal{F} \setminus R$, and let $\mathcal{C}' \in \Sigma_R^n$ satisfy

$$d_H(\mathcal{C}', \mathcal{C}) \leq \min_{Z \in \Sigma_R^n} d_H(Z, \mathcal{C}),$$

and let $\mathcal{C}'' \in \Sigma_{R \cup \{S\}}^n$ satisfy

$$d_H(\mathcal{C}'', \mathcal{C}) \leq \min_{Z \in \Sigma_{R \cup \{S\}}^n} d_H(Z, \mathcal{C}).$$

Then, by the definition of φ ,

$$\begin{aligned} \varphi(R \cup \{S\}, \mathcal{C}) - \varphi(R, \mathcal{C}) &= [d_{\text{opt}} - d_H(\mathcal{C}, \mathcal{C}'')] - [d_{\text{opt}} - d_H(\mathcal{C}, \mathcal{C}')], \\ &= d_H(\mathcal{C}, \mathcal{C}') - d_H(\mathcal{C}, \mathcal{C}''), \\ &= d_H(\mathcal{C}, \mathcal{C}') - \left[d_H(\mathcal{C}, \mathcal{C}') - (d_H(S, \mathcal{C}') - d_H(S, \mathcal{C})) \right], \\ &= d_H(S, \mathcal{C}') - d_H(S, \mathcal{C}), \\ &< (\rho - 1)d_{\text{opt}}. \end{aligned}$$

Since for all $S \in \mathcal{F}$, $d_H(S, \mathcal{C}) \leq d_{\text{opt}}$, we conclude that

$$d_H(S, \mathcal{C}') < (\rho - 1)d_{\text{opt}} - d_H(S, \mathcal{C}) \leq \rho d_{\text{opt}}.$$

□

Theorem 4.1 *For any set \mathcal{F} of strings and any integer $1 \leq r \leq |\mathcal{F}|$, there exists a set $R \subseteq \mathcal{F}$ and a string $\mathcal{C} \in \Sigma_R^n$ such that for all $S \in \mathcal{F}$, $d_H(\mathcal{C}, S) \leq (1 + 1/r)d_{\text{opt}}$ and $|R| = r$.*

Proof. Let \mathcal{C} be an optimal center for \mathcal{F} . Define $R \subseteq \mathcal{F}$, $|R| = r$, such that

$$\varphi(R, \mathcal{C}) \geq \max_{R' \subseteq \mathcal{F}, |R'|=r} (\varphi(R', \mathcal{C})).$$

Suppose $\varphi(R, \mathcal{C}) \geq (1 - 1/r)d_{\text{opt}}$, then there exists a string $\mathcal{C}' \in \Sigma_R^n$ such that $d_H(\mathcal{C}', \mathcal{C}) \leq d_{\text{opt}}/r$. By the triangle inequality, \mathcal{C}' is the desired string.

Suppose $\varphi(R, \mathcal{C}) < (1 - 1/r)d_{\text{opt}}$, then, by the pigeonhole principle, there exists a string $S' \in R$ such that $\varphi(R, \mathcal{C}) - \varphi(R \setminus \{S'\}, \mathcal{C}) < d_{\text{opt}}/r$. If it is true that for all $\mathcal{C}' \in \Sigma_R^n$, there exists a string $S \in \mathcal{F}$ such that $d_H(S, \mathcal{C}') > (1 + 1/r)d_{\text{opt}}$, then, by Lemma 4.3, it is also true that there exists a string $S \in \mathcal{F}$ such that $\varphi(R \cup \{S\}, \mathcal{C}) - \varphi(R, \mathcal{C}) \geq d_{\text{opt}}/r$. This implies

$$\varphi((R \cup \{S\}) \setminus \{S'\}, \mathcal{C}) > \varphi(R, \mathcal{C}),$$

contradicting the definition of R . □

Now we have the information required to solve CLOSEST SUBSTRING in polynomial time if $d \in O(\ln(m))$. By exhaustive sampling, obtain the correct size ϵ^{-1} subset $R \subseteq \mathcal{F}$ indicated in Theorem 4.1. Select an arbitrary string $S \in R$, and for each string $\mathcal{C} \in N_{l,d}(S) \cap \Sigma_R^l$, evaluate \mathcal{C} as a center. The best center tested is guaranteed to have radius within a factor of $1 + \epsilon$ of the optimal radius.

Closest String minimax integer program

Remaining focused on the CLOSEST STRING problem, we assume that the occurrences of a center are known. The case of $d_{\text{opt}} \geq 3\epsilon^{-2} \ln(m)$ can be handled by the technique of randomized rounding and derandomization by conditional probabilities introduced in [74] and [75]. First we formulate the CLOSEST STRING problem as an integer program. Define the set of (indicator) variables

$$\{a_{ij\alpha} : 1 \leq i \leq m, 1 \leq j \leq l, 1 \leq \alpha \leq |\Sigma|\},$$

where $a_{ij\alpha}$ has the value 1 exactly when $S_i[j] = \alpha$, for $\alpha \in \Sigma$. Define the $m \times (l|\Sigma|)$ matrix $A = [a_{ijk}]$, with rows corresponding to the strings $S \in \mathcal{F}$. The rows of A are the constraints particular to each problem instance. A solution is a vector $x = (x_{j\alpha}) \in \{0, 1\}^{l|\Sigma|}$, satisfying the additional constraint that for all j , there is exactly one $\alpha \in \Sigma$ with $x_{j\alpha} = 1$. The formulation is as follows.

$$\begin{aligned} & \text{minimize} && d \\ & \text{subject to} && Ax \leq d \\ & && \sum_{\alpha} x_{j\alpha} = 1, \\ & && x_{j\alpha} \in \{0, 1\}, \text{ for all } j \end{aligned} \tag{4.1}$$

The integer program for CLOSEST STRING is a *minimax* integer program [9]. It is a property of minimax integer programs that while searching for an optimal solution, one need not be concerned with feasibility. Using an algorithm such as Karmarkar's [49], the fractional relaxation of the above integer program can be solved in polynomial time. The resulting solution vector is $\bar{x} \in [0, 1]^{l|\Sigma|}$. Let \bar{d} be the optimal value of the objective function for the relaxation. Following the method of [74], we apply randomized rounding by choosing $x_{j\alpha}$ to represent position j with probability $\bar{x}_{j\alpha}$. In other words, for each j , we select exactly one $\alpha \in \Sigma$ with probability given by $\bar{x}_{j\alpha}$, and set $x_{j\alpha} = 1$. The choice is made independently for each distinct position j , and the choice is mutually exclusive among all $x_{j\alpha}$ for each particular j .

Since, by assumption, the value \bar{d} of the objective function satisfies $\bar{d} > 3\epsilon^{-2} \ln(m)$, a simple application of Chernoff's bound [20] indicates the probability that the randomized rounding procedure fails. Let \mathcal{C} be the center corresponding to the vector x obtained by randomized rounding, then for any $S \in \mathcal{F}$,

$$\Pr \left(d_H(\mathcal{C}, S) > (1 + \epsilon)\bar{d} \right) < \frac{1}{m}.$$

Using Boole's inequality,

$$\Pr \left(\bigvee_{i=1}^m d_H(\mathcal{C}, S_i) > (1 + \epsilon)\bar{d} \right) < \sum_{i=1}^m \Pr \left(d_H(\mathcal{C}, S_i) > (1 + \epsilon)\bar{d} \right) < 1,$$

and we conclude that with positive probability, randomized rounding produces a solution of value at most $(1 + \epsilon)$ times optimal.

The randomized rounding procedure can be fully derandomized using the method of conditional probabilities with a pessimistic estimator as described in [75]. The method uses a pessimistic estimator function to guide a greedy algorithm. The basic

method of pessimistic estimators is described in [63] (Section 5.6). For the specific application of interest here, the pessimistic estimator used in [75] (Section 3) is sufficient.

Estimating the occurrence set

We now return to the CLOSEST SUBSTRING problem. It remains to indicate how to obtain the occurrences of the center in polynomial time. The method used in [61] is based on random sampling, but requires an enumerative stage that depends exponentially on the size of the alphabet. With knowledge of Theorem 4.1, we can modify the method of [61] to be independent of the alphabet size. We incur only a small increase in the error of the performance ratio, and the algorithm remains a polynomial time approximation scheme.

Lemma 4.4 (Sampling Lemma [5]) *Let A be a sequence of numbers taking values from the set $\{0, 1\}$. Let the set Z be a multiset of numbers from A chosen independently at random (with replacement), such that $|Z| = c\epsilon^{-2} \ln(x)$, for some $x \in \mathbb{R}$. Then the sum of members of Z satisfies*

$$\left(\sum_{a \in A} a - \epsilon|Z| \right) \leq \sum_{z \in Z} z \left(\frac{|A|}{|Z|} \right) \leq \left(\sum_{a \in A} a + \epsilon|Z| \right),$$

with probability at least $1 - x^{-c}$.

Proof. Because the elements of Z are chosen randomly with replacement, each $z \in Z$ is a Bernoulli trial with

$$\Pr[z = 1] = \frac{\sum_{a \in A} a}{|A|}.$$

Applying Hoeffding's bound [46],

$$\Pr \left(\sum_{z \in Z} z \left(\frac{|A|}{|Z|} \right) \geq \sum_{a \in A} a + \epsilon|Z| \right) \leq e^{-2\epsilon^2|Z|} < \frac{1}{x^{2c}},$$

and

$$\Pr \left(\sum_{z \in Z} z \left(\frac{|A|}{|Z|} \right) \leq \sum_{a \in A} a - \epsilon|Z| \right) \leq e^{-2\epsilon^2|Z|} < \frac{1}{x^{2c}}.$$

The theorem follows by applying Boole's inequality. \square

To apply the sampling lemma to the CLOSEST SUBSTRING problem, we assume for the moment that we have access to a source of random bits. Fix an instance

$\langle \mathcal{F}, l \rangle$ of CLOSEST SUBSTRING, and a string \mathcal{S} over Σ . We define the variables a_{ijk} with respect to \mathcal{F} , l and \mathcal{S} such that

$$a_{ijk} = \begin{cases} 0 & \text{if } s_{ij}[k] = \mathcal{S}[k] \\ 1 & \text{otherwise.} \end{cases}$$

For each substring s_{ij} define a function $f_{ij} : 2^l \rightarrow \{0, \dots, l\}$ as

$$f_{ij}(X) = \sum_{k \in X} a_{ijk},$$

for all $X \subseteq \{1, \dots, l\}$. Thus $f_{ij}(X) = d_H(\mathcal{S}|_X, s_{ij}|_X)$, for any substring s_{ij} . Now let \mathcal{S} be any length l string over the alphabet of \mathcal{F} , and define the event E_X with respect to \mathcal{F} , l and \mathcal{S} as follows:

$$E_X \equiv \exists s_{ij} \left((1 - \epsilon)d_H(\mathcal{S}, s_{ij}) > f_{ij}(X) \vee f_{ij}(X) > (1 + \epsilon)d_H(\mathcal{S}, s_{ij}) \right).$$

The next lemma follows from a simple application of the sampling lemma.

Lemma 4.5 *Let X be a uniformly random multiset of indexes from $\{1, \dots, l\}$ such that $|X| = 12\epsilon^{-2} \ln(mn)$. Then*

$$\Pr[E_X] \leq \frac{1}{mn}.$$

The result of Theorem 4.1 indicates that for every center \mathcal{C} , there exists a size ϵ^{-1} subset R of the occurrences of \mathcal{C} and a string $\mathcal{C}' \in \Sigma_R^l$ such that \mathcal{C}' is a $(1 + \epsilon)$ -approximate center. Define the set $T_X = \{t_1, \dots, t_m\}$ with respect to \mathcal{F} , l and \mathcal{C}' such that t_i is a substring of $S_i \in \mathcal{F}$, and

$$d_H(\mathcal{C}'|_X, t_i|_X) \leq \min_{1 \leq j \leq n-l+1} d_H(\mathcal{C}'|_X, s_{ij}|_X).$$

Lemma 4.6 *Let X be a uniformly random multiset of positions in $\{1, \dots, l\}$ such that $|X| = 12\epsilon^{-2} \ln(mn)$. With high probability, for all $t_i \in T_X$,*

$$d_H(\mathcal{C}', t_i) \leq (1 + 2\epsilon) \min_{1 \leq j \leq n-l+1} d_H(\mathcal{C}', s_{ij}).$$

Proof. Suppose, for contradiction, that the assertion is false, and let $t_i \in T_X$ be a substring falsifying the assertion. We know that t is not the closest occurrence of \mathcal{C}' that is a substring of S_i . It must be the case that $d_H(\mathcal{C}'|_X, t) \leq d_H(\mathcal{C}', s_{ij})$.

But by Lemma 4.5, the probability of this event is at most $2/(mn)$, which is a contradiction. \square

The set R containing the characters of X is found in polynomial time by testing each possible set. When the correct set R is being tested, as long as $|X| \in O(\ln(mn))$, the characters of $C'|_X$ can be found in polynomial time by testing all strings of length $|X|$ over Σ_R .

Deterministic sampling To remove the assumption that a source of random bits is available, we use the technique mentioned in [59] and [5] of random walks on constant degree expander graphs. The procedure is roughly as follows. Assume that a constant degree expander has been selected, and make a correspondence between the positions of D_R and the vertices of the expander. Since there are explicitly constructible constant degree expanders with efficient algorithms for generating the neighbors of any given vertex, we are not required to explicitly build the graph (see *e.g.* [35]). As the vertices number only $\Theta(|D_R|)$, we may try every vertex as the a starting point for random walks. This enables us to assume the start vertex is selected at random from the stationary distribution (*i.e.* our analysis can assume the most favorable start vertex). For every start vertex, build sets of positions corresponding to the vertices encountered on all walks of length at most $12\epsilon^{-2} \ln(mn)$. Since the degree of the graph is constant, the sets of positions will be polynomial sized. A key result of [39] says that for any sufficiently large subset of the vertices in an expander, on a random walk starting from a random vertex, for any set of vertices the proportion of time spent in that will be a very good approximation to proportional size of the set relative to the size of the graph. More details on expander graphs and deterministic sampling can be found in [63][39][35].

The LARGEALPHABET algorithm

The algorithm LARGEALPHABET is similar to that of [61], but differs in three places where the prior algorithm enumerated over characters of the entire alphabet.

Algorithm 4: Pseudocode for the LARGEALPHABET algorithm.

Input: A set of strings $\mathcal{F} = \{S_1, \dots, S_m\}$, a positive integer $l \leq n$ and an error parameter $\epsilon > 0$.

Output: A string $\mathcal{C} \in \Sigma^l$ such that $\text{radius}_{\mathcal{F}}(\mathcal{C}) \leq (1 + \epsilon)d_{\text{opt}}$.

LARGEALPHABET(\mathcal{F}, l, ϵ)

1. $\mathcal{C} \leftarrow \lambda$
2. **if** SCORE(\mathcal{F}) $\leq 12\epsilon^{-2} \ln(mn)$
3. **for each** $R \subseteq \mathcal{F}$ such that $|R| = \lceil 1/\epsilon \rceil$
4. $S \leftarrow$ (arbitrary string in R)
5. **for each** $X \subseteq D_R, |X| = 12\epsilon^{-2} \ln(mn)$
6. $Z|_{\bar{X}} \leftarrow S|_{\bar{X}}$
7. **for each** string $s \in \Sigma_R^{|X|}$
8. $Z|_X \leftarrow s$
9. **if** SCORE(Z) $<$ SCORE(\mathcal{C}) **then** $\mathcal{C} \leftarrow Z$
10. **if** SCORE(\mathcal{F}) $> 12\epsilon^{-2} \ln(mn)$
11. **for each** $R \subseteq \mathcal{F}$ such that $|R| = \lceil 1/\epsilon \rceil$
12. **for each** $x \in D_R$
13. $K \leftarrow \{X \subseteq D_R : X \text{ is a length } 12\epsilon^{-2} \ln(mn) \text{ walk}\}$
14. **for each** $X \in K$
15. Obtain the set of positions T_X .
16. Let Z be string corresponding to the integral solution to the CLOSEST STRING minimax integer program.
17. **if** SCORE(Z) $<$ SCORE(\mathcal{C}) **then** $\mathcal{C} \leftarrow Z$
18. **output**(\mathcal{C})

Theorem 4.2 *The LARGEALPHABET algorithm is a PTAS for CLOSEST SUBSTRING with time complexity $O((mn)^{c\epsilon^{-2} \ln \epsilon^{-1}})$, where c is an absolute constant not depending on any of the parameters $\{m, n, l, d_{\text{opt}}, |\Sigma|, \epsilon^{-1}\}$.*

Proof. Consider first the case of $d_{\text{opt}} \leq 12\epsilon^{-2} \ln(mn)$. By Theorem 4.1, for some size ϵ^{-1} subset $R \subseteq \mathcal{F}$, the algorithm has a position specific alphabet that is sufficient to construct a $(1 + \epsilon)$ -approximate center. Since the algorithm tests each such subset, it is guaranteed to process a subset R having the desired property. Fix an arbitrary string $S \in R$. Since $d_{\text{opt}} < 12\epsilon^{-2} \ln(mn)$, only $12\epsilon^{-2} \ln(mn)$ positions in \mathcal{C} within D_R may differ from the same positions in S . By observing that $|D_R| \leq 12\epsilon^{-3} \ln(mn)$, we can bound the number of potential centers tested for each

set R by

$$\binom{12\epsilon^{-3} \ln(mn)}{12\epsilon^{-2} \ln(mn)} (1/\epsilon)^{12\epsilon^{-2} \ln(mn)} \leq (e/\epsilon)^{24(\epsilon)^{-2} \ln(mn)} = (mn)^{c\epsilon^{-2} \ln \epsilon^{-1}},$$

(the binomial identity used can be derived using Sterling's formula [40]). Multiplying by another factor of

$$\binom{m}{1/\epsilon} (n-l+1)^{\epsilon^{-1}} \leq (mn)^{\epsilon^{-1}},$$

to account for enumerating all ϵ^{-1} sized subsets R of substrings in \mathcal{F} , evaluation of the SCORE function inside the inner most loop of the algorithm occurs no more than $(mn)^{c\epsilon^{-2} \ln \epsilon^{-1}}$ times, for some absolute constant c . The total complexity for small d_{opt} is therefore $O((mn)^{c\epsilon^{-2} \ln \epsilon^{-1}})$.

For $d_{\text{opt}} > 12\epsilon^{-2} \ln(m)$, the sampling procedure requires $O((mn)^{c\epsilon^{-2} \ln(\alpha)})$ time, where α is the degree of the expander. We multiply by a factor of $(mn)^{c\epsilon^{-2} \ln \epsilon^{-1}}$ to enumerate the possible $\mathcal{C}'|_X$, for each X . Once the sets T_X have been found, $O((mn)^c)$ time is required to perform the linear programming and randomized rounding. Hence, for some absolute constant c , the running time of the LARGEALPHABET algorithm is $O((mn)^{c\epsilon^{-2} \ln \epsilon^{-1}})$. \square

4.1.4 Stronger results for binary strings.

The next result concerns instances of CLOSEST STRING problems, and can be extended to apply to the CLOSEST SUBSTRING problem through the LARGEALPHABET algorithm. The CLOSEST STRING problem with binary alphabet has been studied under the name HAMMING CENTER in [37, 38]. In this section we provide a result that improves on both Theorem 4.1 above and Lemma 2.1 in [59] when dealing with binary strings. Like Lemma 2.1 in [59], the result here gives an upper bound on $|I_R|$. It is a trivial property of binary strings that for any $R \subseteq \mathcal{F}$, $B_R = I_R$: each differing position contains the entire alphabet. Thus by bounding $|I_R|$ we also bound $|B_R|$ and $|G_R|$. Suppose S is a binary string, then define the binary string \bar{S} such that for all positions p , $\bar{S}[p] = 1$ iff $S[p] = 0$.

Lemma 4.7 *Let \mathcal{F} be a set of binary strings with optimal center \mathcal{C} . For any set of positions $X \subseteq \{1, \dots, |\mathcal{C}|\}$, there exists a string $S \in \mathcal{F}$ such that $d_H(S|_X, \mathcal{C}|_X) \leq |X|/2$.*

Proof. Suppose there exists a set of positions $X \subseteq \{1, \dots, n\}$ such that for all $S \in \mathcal{F}$, $d_H(S|_X, \mathcal{C}|_X) > |X|/2$. Then for each string $S \in \mathcal{F}$, $d_H(S|_X, \bar{\mathcal{C}}|_X) < d_H(S|_X, \mathcal{C}|_X)$. Therefore a better center is produced if all positions of X are complemented in \mathcal{C} , contradicting the premise that \mathcal{C} is optimal. \square

Theorem 4.3 *For any set \mathcal{F} of binary strings and any integer $1 \leq r \leq m$, there exists a set $R \subseteq \mathcal{F}$ such that $|B_R| \leq d_{\text{opt}}/(2^{r-1})$ and $|R| \leq r$.*

Proof. We proceed by induction on r . Suppose $r = 2$. Then there exists a string $S \in \mathcal{F}$ such that if S is not an optimal center for \mathcal{F} , there is some $S' \in \mathcal{F}$ such that $d_H(S, S') \geq d_{\text{opt}}$. Let $R = \{S, S'\}$. Then $|B_R| \leq (2d - d_H(S, S'))/2 \leq d_{\text{opt}}/2$.

Suppose there exists a set $R \subseteq \mathcal{F}$ such that $|R| = r - 1$ and $|B_R| \leq d_{\text{opt}}/(2^{r-2})$. By Lemma 4.7, there exists a string $S \in \mathcal{F}$ such that

$$d_H(S|_{B_R}, \mathcal{C}|_{B_R}) \leq \frac{|B_R|}{2} \leq \frac{d}{2^{(r-2)}/2} = \frac{d}{2^{(r-1)}}.$$

Let $R' = R \cup \{S\}$. Then $|B_{R'}| \leq d_{\text{opt}}/(2^{r-1})$. \square

Since a large part of the performance ratio of the algorithm is due to estimating an optimal center based on limited knowledge from some set R of occurrences, the tighter upper bound on information *not* given by some such set immediately implies a better algorithm.

Theorem 4.4 *The LARGEALPHABET algorithm is a polynomial time approximation scheme for the HAMMING CENTER problem, with performance ratio*

$$1 + \frac{1}{2^{r-1}} + \epsilon,$$

where r is the size of the sets R of sampled substrings.

4.1.5 Larger constant sized alphabets

The result of this section is that for any optimal center \mathcal{C} , there is a set R of occurrences of \mathcal{C} such that the number of positions where each member of R differs from \mathcal{C} is exponentially small in $|R|$ when $R \in \Omega(\log m)$. This is a preliminary attempt to extend the analysis of Section 4.1.4 in order to provide a similar bound for any constant sized alphabet.

First we analyze a greedy algorithm for CLOSEST STRING. The greedy nature of the algorithm is due to the fact that it commits to a local improvement at each iteration. The algorithm also uses a lazy strategy that bases each decision on information obtained by examining a restricted portion of the input. The algorithm will be used again in Section 4.2.1. The idea of the algorithm is to read the input strings column by column, and for each column j , assign a character to $\mathcal{C}[j]$ before looking at any column j' such that $j' > j$. Algorithm 5 describes this procedure, named GREEDYANDLAZY, in pseudocode.

Algorithm 5: Pseudocode for the GREEDYANDLAZY algorithm.

Input: A set of strings $\mathcal{F} = \{S_1, \dots, S_m\}$.

Output: An $m(1 - |\Sigma|^{-1})$ -approximate center \mathcal{C} for \mathcal{F} .

GREEDYANDLAZY(\mathcal{F})

1. $\mathcal{C} \leftarrow \lambda$
2. **for** $i \leftarrow 1$ **to** n
3. Let $F_i = \{S_1[1..i], \dots, S_m[1..i]\}$
4. **for each** $\alpha \in \Sigma$
5. $X^\alpha \leftarrow \{S \in F_i : d_H(\mathcal{C}\alpha, S) = \max_{S' \in F_i} d_H(\mathcal{C}\alpha, S')\}$
6. Let $\alpha = S_1[i]$
7. **for each** $\beta \in \Sigma$
8. **if** $|X^\beta| < |X^\alpha|$
9. $\alpha = \beta$
10. $\mathcal{C} \leftarrow \mathcal{C}\alpha$
11. **output**(\mathcal{C})

Lemma 4.8 *The greedy and lazy algorithm for CLOSEST SUBSTRING is a $m(1 - |\Sigma|^{-1})$ -approximation algorithm.*

Proof. Consider the number of iterations required to guarantee that each $S \in \mathcal{F}$ matches \mathcal{C} in at least one position. Let J_i be the set of strings that do not match any position of \mathcal{C} after the i^{th} iteration, then

$$J_{i+1} \leq \left(\frac{|\Sigma| - 1}{|\Sigma|} \right) J_i \leq \exp(-1/|\Sigma|) J_i.$$

Let x be the number of iterations required before all members of \mathcal{F} match \mathcal{C} in at least one position. A sufficient condition on the value of x is given by the following

inequality:

$$\frac{1}{m} > \exp\left(-\frac{x}{|\Sigma|}\right).$$

Hence, for any ϵ bounded above 0, after $x = |\Sigma| \ln m + \epsilon$ iterations, each member of \mathcal{F} matches \mathcal{C} in at least one position. After the final iteration, the total distance from \mathcal{C} to any member of \mathcal{F} is at most $n - n/(|\Sigma| \ln m)$. The optimal distance is at least n/m , otherwise some positions are identical in \mathcal{F} (and thus should not be considered). Therefore the performance ratio of GREEDYANDLAZY is

$$\frac{n - n/(|\Sigma| \ln m)}{n/m} \leq \frac{m}{\ln m} \left(1 - \frac{1}{|\Sigma|}\right).$$

□

Corollary 4.1 *Let \mathcal{F} be a set of strings with optimal center \mathcal{C} . For all sets of positions $X \subseteq \{1, \dots, |\mathcal{C}|\}$, there exists a string $S \in \mathcal{F}$ such that $d_H(S|_X, \mathcal{C}|_X) \leq |X| - |X|/\ln m$.*

The result of Corollary 4.1 is an extension of Lemma 4.7 to any constant sized alphabet, and can be used similarly to obtain a performance ratio for CLOSEST STRING and CLOSEST SUBSTRING.

Theorem 4.5 *For any set \mathcal{F} of strings and any integer $1 \leq r \leq m$, there exists a set $R \subseteq \mathcal{F}$ such that $|R| = r$ and*

$$|B_R| \leq d \exp\left(\frac{1-r}{|\Sigma| \ln m}\right).$$

Proof. We proceed by induction on r with base case $r = 2$. For some $S \in \mathcal{F}$, $d_H(S, \mathcal{C}) = d_{\text{opt}}$. Define the set of positions $X = \{i : S[i] \neq \mathcal{C}[i]\}$ and note that $|X| \leq d_{\text{opt}}$. By Lemma 4.8, some $S' \in \mathcal{F}$ must match $d_{\text{opt}}/(|\Sigma| \ln m)$ positions of X . Let $R = \{S, S'\}$, then

$$B_R \leq d(1 - 1/(|\Sigma| \ln m)) \leq d \exp(-1/\ln m).$$

Suppose there exists a set $R \subseteq \mathcal{F}$ such that $|R| = r - 1$ and $|B_R| \leq d \exp((2 - r)/(|\Sigma| \ln m))$. By Lemma 4.8, there exists a string $S \in \mathcal{F}$ such that

$$d_H(S|_{B_R}, \mathcal{C}|_{B_R}) \leq \frac{|B_R|}{\exp(1/(|\Sigma| \ln m))} \leq \frac{d}{\exp((r-1)/(|\Sigma| \ln m))}.$$

Let $R' = R \cup \{S\}$, then $|B_{R'}| \leq d \exp((1 - r)/(|\Sigma| \ln m))$. □

4.2 Approximating Alternate Objectives

Though more attention has been given to the objective of minimizing d , the CLOSEST SUBSTRING problem can be formulated with other objectives. Three such objectives are analyzed here; each providing a more natural abstraction of certain biological applications than CLOSEST SUBSTRING. The different objectives are:

1. Maximize $l - d$, the *similarity* between a length l center and its set of occurrences.
2. Maximize l , the length of the center, while maintaining a maximum distance of d between the center and any occurrence.
3. Maximize m , the number of strings containing an occurrence of some length l center having maximum distance d .

We formally define different problems correspond to the different objective functions. We refer to the problem that is complementary to CLOSEST SUBSTRING as MAX CLOSEST SUBSTRING.

MAX CLOSEST SUBSTRING

Instance: A set $\mathcal{F} = \{S_1, \dots, S_m\}$ of strings over alphabet Σ such that $|S_i| = n$, integer l , ($1 \leq l \leq n$).

Objective: Maximize $l - d_H(\mathcal{C}, s_i)$, such that $\mathcal{C} \in \Sigma^l$ and s_i is a substring of S_i , ($1 \leq i \leq m$).

We also give results concerning the next two problems, which require the value of d to be specified as part of the input.

LONGEST COMMON APPROXIMATE SUBSTRING

Instance: A set $\mathcal{F} = \{S_1, \dots, S_m\}$ of strings over alphabet Σ such that $|S_i| = n$, integer d , ($1 \leq d < n$).

Objective: Maximize $l = |\mathcal{C}|$, $\mathcal{C} \in \Sigma^*$, such that $d_H(\mathcal{C}, s_i) \leq d$ and s_i is a substring of S_i , ($1 \leq i \leq m$).

MAXIMUM COVERAGE APPROXIMATE SUBSTRING

Instance: A set $\mathcal{F} = \{S_1, \dots, S_m\}$ of strings over alphabet Σ such that $|S_i| = n$, integers d and l , ($1 \leq d < l \leq n$).

Objective: Maximize $|\mathcal{F}'|$, $\mathcal{F}' \subseteq \mathcal{F}$, such that for some $\mathcal{C} \in \Sigma^l$ and for all $S_i \in \mathcal{F}'$, there exists a substring s_i of S_i such that $d_H(\mathcal{C}, s_i) \leq d$.

Throughout this section, when discussing the different versions of closest substring, the values of d , l and m may refer to optimal values of objective functions or values specified as part of the input. Use of these symbols will be in accordance with their use in the formal statement of whatever problem is being discussed.

4.2.1 The similarity aspect

Often the distinction between minimizing a distance and maximizing a similarity is of no consequence; such is not the case for CLOSEST SUBSTRING. To minimize a distance function often implies that the optimal value (*i.e.* the minimum) is small. For CLOSEST SUBSTRING, this means that optimal center strings are expected to have a small number of mismatches with respect to their occurrences within the strings. In cases where hybridization is important, it may be more natural to treat the task of finding patterns as one of maximizing the number of possible base pairings. Here it is shown that this difference is more than semantic.

We use a gap preserving reduction from SET COVER to show inapproximability for MAX CLOSEST SUBSTRING. Lund and Yannakakis [60], with a reduction from LABEL COVER to SET COVER, showed that SET COVER could not be approximated in polynomial time with performance ratio better than $(\log |\mathcal{B}|)/4$ (where \mathcal{B} is the base set) unless $\text{NP} = \text{DTIME}(2^{\text{poly}(\log n)})$, *i.e.* deterministic quasi-polynomial time. A result of Raz and Safra [76] indirectly strengthened the conjecture; SET COVER is now known to be NP-hard to approximate with ratio better than $(\log |\mathcal{B}|)/4$.

SET COVER

Instance: A set \mathcal{B} of elements to be covered and a collection of sets \mathcal{L} such that $\mathcal{L}_i \subseteq \mathcal{B}$, $(1 \leq i \leq |\mathcal{L}|)$.

Objective: Minimize $|R|$, $R \subseteq \mathcal{L}$, such that $\cup_{j=1}^{|R|} R_j = \mathcal{B}$.

Let $I = \langle \mathcal{B}, \mathcal{L} \rangle$ be an instance of SET COVER. The reduction constructs, in polynomial time, a corresponding instance $I' = \langle \mathcal{F}, l \rangle$ of MAX CLOSEST SUBSTRING. Additionally, for all $\rho > 1$, there exists a $\rho' > 1$ such that a solution to I with a ratio of ρ can be obtained in polynomial time from a solution to I' with ratio ρ' .

The alphabet The strings of \mathcal{F} are composed of characters from the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$. The characters of Σ_1 are referred to as *set characters*, and identify sets in \mathcal{L} . The characters of Σ_2 are referred to as *element characters* and are in one-to-one correspondence with elements of the base set \mathcal{B} .

$$\begin{aligned}\Sigma_1 &= \{p_i : 1 \leq i \leq |\mathcal{L}|\}, \\ \Sigma_2 &= \{u_i : 1 \leq i \leq |\mathcal{B}|\}.\end{aligned}$$

Substring gadgets The strings of \mathcal{F} are made up of two types of substring gadgets. We use the function f , defined below, to ensure that the substring gadgets are sufficiently large. The gadgets are defined as follows:

$$\begin{aligned}\text{Subset Selectors: } & \langle \text{set}(i) \rangle = p_i^{f(|\mathcal{B}|)} \\ \text{Separators: } & \langle \text{separator}(j) \rangle = u_j^{f(|\mathcal{B}|)}\end{aligned}$$

The reduction The string set \mathcal{F} contains $|\mathcal{B}|$ strings, corresponding to the elements of \mathcal{B} . For each $j \in \mathcal{B}$, let $L_j \subseteq \mathcal{L}$ be the subfamily of sets containing the element j . Then define the string

$$S_j = \prod_{q \in L_j} \langle \text{set}(q) \rangle \langle \text{separator}(j) \rangle.$$

The function $f : \mathbb{N} \mapsto \mathbb{N}$ must be defined. It is necessary for f to have the property that for all positive integers $x < |\mathcal{B}|$,

$$\left\lceil \frac{f(|\mathcal{B}|)}{x} \right\rceil > \left\lceil \frac{f(|\mathcal{B}|)}{x+1} \right\rceil.$$

It is straightforward to check that $f(x) = x^2$ has this property. The importance of this property is explained following Lemma 4.9. The maximum length of any member of \mathcal{F} is $n = 2|\mathcal{L}||\mathcal{B}|^2$, the size of \mathcal{F} is $m = |\mathcal{B}|$, the length of the center is $l = f(|\mathcal{B}|) = |\mathcal{B}|^2$ and the alphabet size is $|\Sigma| = |\mathcal{L}| + |\mathcal{B}|$. We call any partition of \mathcal{F} whose equivalence relation is the property of having an exact common substring a *substring induced partition*. For any two occurrences s, s' of a center, we call s and s' disjoint if for all $1 \leq q \leq |s|$, $s[q] \neq s'[q]$. Observe that the maximum distance to an optimal center, for any set of disjoint occurrences, increases with the size of the set.

Lemma 4.9 *Let F be a set of occurrences of an optimal center \mathcal{C} such that $|F| = k$. If for each pair $s, s' \in F$, $d_H(s, s') = l$, then there exists $s \in \mathcal{F}$ such that $l - d_H(\mathcal{C}, s) = \lceil l/k \rceil$.*

Proof. There are l total positions and for any position p , there is a unique $s \in F$ such that $s[p] = \mathcal{C}[p]$. If some $s \in F$ had $l - d_H(\mathcal{C}, s) < \lceil l/k \rceil$, then the center \mathcal{C} would not be optimal, as a better center can be constructed by taking position symbols evenly from the k occurrences. If all $s \in F$ have $l - d_h(\mathcal{C}, s) > \lceil l/k \rceil$, then the total number of matches exceeds l , some pair of matches would have the same position, and thus some pair $s, s' \in F$ have $d_H(s, s') < l$. \square

The significance of our definition for f is apparent from the above proof. It is essential that, under the premise of Lemma 4.9, values of k (the number of distinct occurrences of a center) can be distinguished based on the maximum distance from any occurrence to the optimal center. In essence, f must be sufficiently large that the substrings behave as continuous, rather than discrete, objects.

Lemma 4.10 SET COVER \leq_{GP} MAX CLOSEST SUBSTRING.

Proof. Suppose the optimal cover R for $\langle \mathcal{B}, \mathcal{L} \rangle$ has size less than or equal to c . Construct string \mathcal{C} of length $|\mathcal{B}|^2$ as follows. To the positions in \mathcal{C} , assign in equal amounts the set characters representing members of R . Then \mathcal{C} is a center for \mathcal{F} with maximum similarity $\lceil |\mathcal{B}|^2/c \rceil$.

Suppose $|R| > c$. Let \mathcal{F}' be the largest subset of \mathcal{F} having a substring induced c -partition. By the reduction, since $|R| > c$, $\mathcal{F}' \neq \mathcal{F}$. Let S be any string in $\mathcal{F} \setminus \mathcal{F}'$. By Lemma 4.9, any optimal center for \mathcal{F}' must have maximum similarity $\lceil |\mathcal{B}|^2/c \rceil$, and therefore has at least $\lceil |\mathcal{B}|^2/c \rceil$ characters from a substring of some string in \mathcal{F}' . But the occurrence in S is disjoint from the occurrences in \mathcal{F}' , forcing the optimal center to match an equal number of positions in more than c disjoint occurrences. Hence the optimal center matches at most $\lceil |\mathcal{B}|^2/(c+1) \rceil < \lceil |\mathcal{B}|^2/c \rceil$ characters in each occurrence. The gap preserving property of the reduction follows from the fact that $\lceil |\mathcal{B}|^2/c \rceil$ is a decreasing function of c . \square

Theorem 4.6 MAX CLOSEST SUBSTRING is not approximable within $(\log m)/4$ in polynomial time unless P=NP.

Proof. The theorem follows from the fact that the NP-hard ratio for MAX CLOSEST SUBSTRING remains identical to that of the source problem SET COVER. \square

As MAX CLOSEST STRING is the complementary maximization version of CLOSEST SUBSTRING, and there is a bijection between feasible solutions to the complementary problems that preserves the order of solution quality, this reduction also applies to CLOSEST SUBSTRING. The form of the hard performance ratio for CLOSEST SUBSTRING provides evidence that the two separate sources of error, $1/(2r-1)$ and ϵ , are necessary in the PTAS of [59].

Theorem 4.7 CLOSEST SUBSTRING *cannot be approximated with performance ratio $1 + \frac{1}{2m}$ in polynomial time unless P=NP.*

Proof. Since the NP-hard ratio for SET COVER is $\rho = (1/4) \log |\mathcal{B}|$, the NP-hard ratio obtained for CLOSEST SUBSTRING in the above reduction is

$$\begin{aligned} \rho' &= \frac{c\rho-1}{c\rho-\rho} \\ &= 1 + \left(\frac{\rho-1}{\rho}\right) \cdot \left(\frac{1}{c-1}\right) \\ &\geq 1 + \frac{1}{O(m)}. \end{aligned}$$

□

With respect to Theorem 4.6, the bound of $\Omega(\log m)$ is asymptotically tight if the alphabet size is held constant.

Theorem 4.8 *The GREEDYANDLAZY algorithm is a $O(|\Sigma| \ln m)$ -approximation algorithm for MAX CLOSEST SUBSTRING.*

Proof. The result follows directly from Lemma 4.8. The precise performance ratio given by the GREEDYANDLAZY algorithm is

$$|\Sigma| \ln m,$$

which is $O(\log m)$ for constant sized alphabet. □

4.2.2 The quorum aspect

The objective of maximizing the number of sequences that contain an occurrence of the pattern has been found to be useful. Vanet et al. [89] used an algorithm of Sagot [78] to infer regulatory elements on a genomic scale. The algorithms of Sagot includes a *quorum* parameter, which specifies how many among a set of sequences

must contain an occurrence of a pattern. The quorum parameter is important for pattern discovery on a genomic scale, since regions of interest (genes or regulatory regions) may number in the thousands. It is not expected that all of these regions are connected by some pattern; the goal of genomic scale analysis is often to organize regions into families according to the patterns they contain.

Here we show that the MAXIMUM COVERAGE APPROXIMATE SUBSTRING problem is APX-Hard. An incorrect reduction in [61] claimed an NP-hard ratio of $O(n^\epsilon)$, $\epsilon = \frac{1}{4}$, for MAXIMUM COVERAGE APPROXIMATE SUBSTRING when $l = n$ and $|\Sigma| = 2$. Its error resulted from applying Theorem 5 of [54], proven only for alphabet size at least three, to binary strings. Hardness of approximation for the general problem is shown here by a reduction from MAXIMUM COVERAGE.

MAXIMUM COVERAGE

Instance: A set \mathcal{B} of elements to be covered and a collection of sets \mathcal{L} such that $\mathcal{L}_i \subseteq \mathcal{B}$, ($1 \leq i \leq |\mathcal{L}|$), a positive integer k .

Question: Maximize $|B|$, $B \subseteq \mathcal{B}$, such that $B = \cup_{j=1}^k \mathcal{L}_j$, where $\mathcal{L}_j \in \mathcal{L}$.

Given an instance $\langle \mathcal{B}, L, k \rangle$ of MAXIMUM COVERAGE, we construct an instance $\langle \mathcal{F}, l, d \rangle$ of MAXIMUM COVERAGE APPROXIMATE SUBSTRING where $m = |\mathcal{B}|$, $l = k$, $d = k - 1$ and $n \leq k|\mathcal{L}|$. The construction of \mathcal{F} is similar to the construction used when reducing from SET COVER to CLOSEST SUBSTRING in Section 4.2.1; unnecessary parts are removed.

The alphabet The strings of \mathcal{F} are composed of characters from the alphabet Σ . The characters of Σ correspond to the sets $\mathcal{L}_i \in \mathcal{L}$ that can be part of a cover, so $\Sigma = \{x_i : 1 \leq i \leq |\mathcal{L}|\}$.

The reduction The string set $\mathcal{F} = \{S_1, \dots, S_{|\mathcal{B}|}\}$ contains strings corresponding to the elements of \mathcal{B} . For each $j \in \mathcal{B}$, let $L_j \subseteq \mathcal{L}$ be the subfamily of sets containing the element j . Letting product notation refer to concatenation, for each $j \in \mathcal{B}$, define the string

$$S_j = \prod_{x_i \in L_j} x_i^k.$$

Set $d = k - 1$ and $l = k$. We seek to maximize the number of strings in \mathcal{F} containing occurrences of some center \mathcal{C} .

Lemma 4.11 MAXIMUM COVERAGE \leq_{GP} MAXIMUM COVERAGE APPROXIMATE SUBSTRING.

Proof. Suppose $\langle \mathcal{L}, \mathcal{B}, k \rangle$ is an instance of MAXIMUM COVERAGE with a solution set $R \subset \mathcal{L}$, such that $|R| = k$ and R covers $b \leq |\mathcal{B}|$ elements. Then there is a center \mathcal{C} for \mathcal{F} of length $l = k$ that has distance at most $d = k - 1$ from a substring of b strings in \mathcal{F} . Let the k positions in \mathcal{C} be assigned characters representing the k sets in the cover, *i.e.* for each $x_i \in R$, there is a position p such that $\mathcal{C}[p] = x_i$. All b members of \mathcal{F} corresponding to those covered elements in \mathcal{B} contain a substring matching at least one character in \mathcal{C} , and mismatch at most $k-1$ characters. Suppose one cannot obtain a k cover with ratio better than ρ . Then one cannot obtain a center for \mathcal{F} that occurs in more than b/ρ strings of \mathcal{F} , so the hard ratio is $\rho' = \frac{b}{b/\rho} = \rho$. \square

Theorem 4.9 MAXIMUM COVERAGE APPROXIMATE SUBSTRING *cannot be approximated with performance ratio better than $1 + 1/e - \epsilon$, for any $\epsilon > 0$, unless P=NP.*

Proof. It was shown in [31] that the NP-hard ratio for MAXIMUM COVERAGE is $1 + 1/e - \epsilon$. This result, combined with Lemma 4.11 proves the theorem. \square

Let $\langle \mathcal{F}, l, d \rangle$ be an instance of MAXIMUM COVERAGE APPROXIMATE SUBSTRING, with optimal center \mathcal{C} . For each string s , of length l , define the two sets

$$R_s^i = \left\{ s' : \exists S \in \mathcal{F}, (s' \in S^l \wedge d_H(s, s') \leq d + i) \right\},$$

$$G_s^i = \left\{ S \in \mathcal{F} : \exists s' \in R_s^i, (s' \in S^l) \right\}.$$

Notice that if $s \in G_{\mathcal{C}}^0$, then $|G_s^d| \geq |G_{\mathcal{C}}^0|$. The algorithm is as follows. For every substring s of a member of \mathcal{F} , the algorithm attempts to construct a center from s . The center is constructed iteratively by changing d positions in s . At each iteration, the position that gets changed is the one that will have the smallest effect on $|G_s|$. Both R_s and G_s get re-evaluated after the change. For each i from 1 to d , obtaining R_s and G_s , identify a position to change in s and change the position, modifying s .

Algorithm 6: Pseudocode for the GREEDYMAXIMUMCOVERAGE algorithm. The symbol λ indicates the empty string. When a string S is used as the set of substrings of S , those substrings are assumed to have length l .

Input: A set of strings $\mathcal{F} = \{S_1, \dots, S_m\}$ and an integer d .

Output: A valid center \mathcal{C} for \mathcal{F} .

GREEDYMAXIMUMCOVERAGE(\mathcal{F}, l, d)

1. $\mathcal{C} \leftarrow \lambda, x \leftarrow 0$
2. **for each** $S \in \mathcal{F}$
3. **for each** $s \in S$
4. **if** $|G_s^0| > x$ **then** $\mathcal{C} \leftarrow s, x \leftarrow |G_s^0|$
5. **for** $i \leftarrow 1$ **to** d
6. Substitute $s[p] \leftarrow \alpha$ to minimize $|G_s^{i-1}| - |G_s^i|$.
7. **if** $|G_s^i| > x$ **then** $\mathcal{C} \leftarrow s, x \leftarrow |G_s^i|$
8. **output**(\mathcal{C})

Theorem 4.10 GREEDYMAXIMUMCOVERAGE is a $|\Sigma|^d$ -approximation algorithm.

Proof. For some s , the set G_s is at least as large as $G_{\mathcal{C}}$. At each iteration, the modification to s removes at most a fraction of $(|\Sigma| - 1)/|\Sigma|$ members from the covered set G_s . The coverage obtained by the greedy algorithm is then $1/(|\Sigma|^d)$ times optimal. It follows that the performance ratio is $|\Sigma|^d$. \square

Note that the reduction shows hardness for the general version of the problem, and leaves open the restricted case of $l = n$ with $|\Sigma| = 2$. A different type of approximation is possible. The idea of an ϵ relaxed decision procedure was employed by Plotkin et al. [73] in the context of approximating polynomial time solvable packing and covering problems. An ϵ -relaxed decision procedure either finds a $1 + \epsilon$ approximate solution, or correctly concludes that no exact solution exists. We change the definition slightly here.

Definition 4.2.1 A ρ -relaxed decision procedure is an optimization algorithm that finds a solution with performance ratio ρ , or correctly concludes that no exact solution exists.

We look at the restricted problem where $l = n$.

Theorem 4.11 The column majority heuristic is a $(d + 1)$ -relaxed decision procedure for MAXIMUM COVERAGE APPROXIMATE SUBSTRING when $l = n$.

Proof. Suppose the instance is feasible, *i.e.* there is a string \mathcal{C} of length $l(= n)$ such that for all $S \in \mathcal{F}$, $d_H(\mathcal{C}, S) \leq d$. Then the column majority string will have at most md mismatches with respect to members of \mathcal{F} (this follows from Lemma 3.4). The number of strings mismatching more than d positions is maximized when each of these mismatches exactly $d + 1$ positions:

$$\frac{m - md/(d + 1)}{m} = \frac{1}{d + 1}.$$

Therefore at least $m/(d + 1)$ strings match the column majority string in $l - d$ positions.

Suppose the column majority heuristic concludes that no center with radius d exists for \mathcal{F} . Then the column majority string has more than md total mismatches with respect to members of \mathcal{F} , and the optimal center has distance $> d$. \square

4.2.3 The length aspect

Two of the most popular computational tools used by biologists are the BLAST (Basic Local Alignment Search Tool) [3][2] and FASTA [70] programs. The algorithmic foundation of these programs is to find short exactly matching patterns, under the assumption that some of them are likely to be contained in longer approximate patterns. These exact matches, called hot spots or hits, are then extended in both directions until the “score” (whatever score is being used) no longer increases.

It is plausible that such a strategy may perform well in solving CLOSEST SUBSTRING type problems. The complexity of a similar *extending* strategy is analyzed in this section. The formal problem dealt with here is LONGEST COMMON APPROXIMATE SUBSTRING, and seeks to maximize the length of a center that is within a specified distance from each string in the problem instance.

That a feasible solution always exists can be seen by considering the case of a single character, since the problem is defined with $d > 0$. Here we show that a simple algorithm always produces a valid center that is at least half the optimal length. A valid center is any string that has distance at most d from at least one substring of each string in \mathcal{F} . The algorithm simply evaluates each substring of members of \mathcal{F} and tests them as centers. The following procedure EXTENDOCURRENCE accomplishes this with a time complexity of $\Theta(m^2n^3)$.

Algorithm 7: Pseudocode for the EXTENDOCURRENCE algorithm.

Input: A set of strings $\mathcal{F} = \{S_1, \dots, S_m\}$ and an integer d .

Output: A valid center \mathcal{C} for \mathcal{F} .

EXTENDOCURRENCE(\mathcal{F}, d)

1. Let \mathcal{C} be λ , the empty string
2. **for each** string S_i in \mathcal{F}
3. **for each** substring c of S_i
4. **if** VALID(c, \mathcal{F}, d) = true **then** $\mathcal{C} \leftarrow c$
5. **output**(\mathcal{C})

Theorem 4.12 EXTENDOCURRENCE is a 2-approximation algorithm for LONGEST COMMON APPROXIMATE SUBSTRING.

Proof. Let \mathcal{C} be the optimal center for \mathcal{F} . For each $S_i \in \mathcal{F}$, let s_i be the occurrence of \mathcal{C} from S_i ; observe that $|s_i| = |\mathcal{C}|$. Define s_i^1 as the substring of s_i consisting of the first $|\mathcal{C}|/2$ positions of s_i , and s_i^2 as the substring consisting of the remaining positions. Similarly, define \mathcal{C}^1 and \mathcal{C}^2 as the first and last half of \mathcal{C} . For $x \in \{1, 2\}$, define s^x as a string satisfying

$$s^x = s_i^x \Rightarrow d_H(s_i^x, \mathcal{C}^x) \leq \min_{s_j^x, j \neq i} d_H(s_j^x, \mathcal{C}^x).$$

Define s such that

$$s = \begin{cases} s^1 & \text{if } d(s^1, \mathcal{C}^1) \leq d(s^2, \mathcal{C}^2), \\ s^2 & \text{otherwise.} \end{cases}$$

Note that $d_H(s, \mathcal{C}^x) \leq d/2$, for some $x \in \{1, 2\}$. Suppose, for contradiction, that s is not a valid center. Assume, without loss of generality, that $s = s_i^1$ for some i . Then there is some s_j^1 such that $d_H(s, s_j^1) > d$. Since $d_H(s, \mathcal{C}^1) = d/2 - y$ for some $1 \leq y \leq d/2$, by the triangle inequality $d_H(s_j^1, \mathcal{C}^1) \geq d/2 + y + 1$. This implies that $d_H(s_j^2, \mathcal{C}^2) \leq d/2 - y - 1 < d_H(s, \mathcal{C}^1)$, contradicting the definition of s . Hence s is a valid center. \square

The performance ratio of 2 is optimal unless P=NP. We use a transformation from the VERTEX COVER decision problem that introduces a gap in the objective function.

VERTEX COVER

Instance: A graph $G = (V, E)$ and a positive integer k .

Question: Does G have a vertex cover of size at most k , i.e., a set of vertices $V' \subseteq V$, $|V'| \leq k$, such that for each edge $(u, v) \in E$, at least one of u and v belongs to V' ?

Suppose for some graph G , we seek to determine if G contains a vertex cover of size k . We construct an instance of LONGEST COMMON APPROXIMATE SUBSTRING with $|E|$ strings corresponding to the edges of G . The intuition behind the reduction is that an occurrence of the center in each string corresponds to the occurrence of a cover vertex in the corresponding edge. Before giving values of n and d , we describe the gadgets used in the reduction.

The alphabet The string alphabet is $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \{A\}$. We refer to these as vertex characters (Σ_1), unique characters (Σ_2), and the alignment character (A), where $\Sigma_1 = \{v_i : 1 \leq i \leq |V|\}$ and $\Sigma_2 = \{u_{ij} : (i, j) \in E\}$.

Substring gadgets We next describe the two “high level” component substrings used in the construction. The function f is any arbitrarily large polynomial function of $|G|$.

$$\text{Vertex Selectors: } \langle \text{vertex}(x, i, j, z) \rangle = A^{f(k)} u_{ij}^{(z-1)} v_x u_{ij}^{(k-z)} A^{f(k)}$$

$$\text{Separators: } \langle \text{separator}(i, j) \rangle = u_{ij}^{3f(k)}$$

The reduction We construct \mathcal{F} as follows. For any edge $(i, j) \in E$:

$$S_{ij} = \prod_{1 \leq z \leq k} \langle \text{vertex}(i, i, j, z) \rangle \langle \text{separator}(i, j) \rangle \langle \text{vertex}(j, i, j, z) \rangle \langle \text{separator}(i, j) \rangle$$

The length of each string is then $n = k(10f(k) + 2k)$. The threshold distance is $d = k - 1$.

Theorem 4.13 LONGEST COMMON APPROXIMATE SUBSTRING *cannot be approximated in polynomial time with performance ratio better than $2 - \epsilon$, for any $\epsilon > 0$, unless P=NP.*

Proof. For any set of strings \mathcal{F} , constructed as in the reduction from an instance of VERTEX COVER, there is an exact common substring of length $f(k)$ corresponding to the $f(k)$ repeats of the alignment character A . Suppose there is a size k cover for the source instance of VERTEX COVER. Construct a center \mathcal{C} for \mathcal{F} as follows.

Assign the alignment character A to the first $f(k)$ positions in \mathcal{C} . To positions $f(k) + 1$ through $f(k) + k$, assign the characters corresponding to the vertices in the vertex cover. These may be assigned in any order. Finally, assign the alignment character A to the remaining $f(k)$ positions of \mathcal{C} . Each string in \mathcal{F} contains a substring that matches $2f(k) + 1$ positions in \mathcal{C} , so \mathcal{C} is a valid center.

In case there is no k cover for the source instance of VERTEX COVER, then for any length $f(k) + k$ string there will be some $S \in \mathcal{F}$ that mismatches k positions. As f can be any arbitrarily large polynomial function of k , the NP-hard performance ratio is

$$\frac{2f(k) + k}{f(k) + k} \geq 2 - \epsilon,$$

for any constant $\epsilon > 0$. \square

4.3 Connection with Parameterized Complexity

4.3.1 Parameterized approximation

The parameterized analysis of Chapter 3 did not prove any difference between the complexity of $\text{CAS}(l)$ and the complexity of $\text{CAS}(d)$. Using the results of the present section, a simple algorithm for approximating CLOSEST SUBSTRING is possible *when parameterized with d* . While a PTAS was given in [59] for CLOSEST SUBSTRING under constant alphabet size, and extended in this thesis to handle an arbitrary alphabet size, allowing the fixed parameter d simplifies the algorithm significantly.

Fixing d is unusual because d is the objective function for CLOSEST SUBSTRING; objective function values, by definition, cannot be specified as part of an instance of an optimization problem. There are two reasons why, in the case of CLOSEST SUBSTRING, it is reasonable to fix the value of the objective function. The first is the fact that the objective is to *minimize* d , which indicates that better solutions will not require more time than solutions of lower quality. The second reason is that tight bounds on the value of d can be obtained easily using Lemma 2.1, which allows the value of d in the complexity algorithm to remain less than twice the optimal value.

An important article by Cesati and Trevisan [19] established the notion of an *efficient polynomial time approximation scheme* or EPTAS. The definition of an EPTAS treats the performance ratio of an algorithm as a parameter. If the solution produced by an EPTAS is an r -approximate solution, the time complexity of the

EPTAS must be of the form $O(f(r)n^c)$, where f is an arbitrary function, and c is a constant. Unfortunately the algorithm of this section is not an EPTAS, and the question of whether such an algorithm exists remains open.

Initially, the algorithm scans the input to obtain a 2-approximate solution, which removes the need in subsequent steps for d to assume values greater than twice the optimal value. Then the algorithm builds a center by selecting each possible set of r occurrences, of which there are $O((mn)^r)$. Given a set of possible occurrences, the positions that are identical within the occurrences are assigned to corresponding positions in \mathcal{C} . There are at most rd remaining positions, and the algorithm enumerates all strings that may be assigned to those positions, and then tests the constructed center. The algorithm is called BOUNDENUMERATION, and is described in pseudocode in Algorithm 8.

Algorithm 8: Pseudocode for the BOUNDENUMERATION algorithm.

Input: A set of strings $\mathcal{F} = \{S_1, \dots, S_m\}$ and two integers $r < l$.

Output: A $(1 + 1/r)$ -approximate center \mathcal{C} for \mathcal{F} .

BOUNDENUMERATION(\mathcal{F}, r, l)

1. **for** $\hat{d} \leftarrow 1$ **to** d
2. **for each** $R = \{S_{i_1}, \dots, S_{i_r}\} \subseteq \mathcal{F}$ such that $|R| = r$
3. **for each** $R' = \{s_1, \dots, s_r\} \in S_{i_1} \times \dots \times S_{i_r}$
4. **for each** $Z \in \Sigma_{R'} : \forall s_i \in R', d_H(s_i, Z) \leq \hat{d}$
5. **if** SCORE(Z, \mathcal{F}) \leq SCORE(\mathcal{C}, \mathcal{F}) **then** $\mathcal{C} \leftarrow Z$
6. **output**(\mathcal{C})

Theorem 4.14 *The algorithm BOUNDENUMERATION runs in $O(de^d r^{2d} (mn)^{r+1})$ and guarantees a $(1 + 1/r)$ -approximate solution.*

Proof. The first step requires $\Theta((mn)^2)$ time, but this is additive and does not have an asymptotic effect. The number of sets of possible occurrences that will be tested is at most $\binom{m}{r} (n - l + 1)^r \in O((mn)^r)$. For each such set, the number of centers constructed and scored is

$$\binom{rd}{d} |\Sigma_{R'}|^d \leq e^d r^{2d}.$$

Obtaining a score for each center introduces another factor of $\Theta(mn)$. The entire procedure is repeated for each value of \hat{d} until d is reached. The performance ratio of the algorithm is a consequence of Theorem 4.1. \square

4.3.2 Hardness by parameterized reductions

Parameterized complexity can be used to gain additional insight into the approximation properties of CLOSEST SUBSTRING.

Theorem 4.15 *There is no approximation scheme for CLOSEST SUBSTRING running in time $O(f(m, l, 1/\epsilon)n^{c_1}|\Sigma|^{c_2})$ unless $FPT=W[1]$ and there is no approximation scheme for CLOSEST SUBSTRING running in time $O(f(l, 1/\epsilon)n^{c_1}|\Sigma|^{c_2})$ unless $FPT=W[2]$.*

Proof. Suppose the theorem does not hold. Then select $\epsilon = 1/(2d)$, providing an error of less than one unit (no error). This would imply that $CAS(l)$ and $CAS(m, l)$ are in FPT. As the problems are hard for $W[2]$ and $W[1]$ respectively, these classes would collapse to FPT. \square

4.4 Summary and Open Problems

This chapter focused on the pattern discovery problem as the problem of finding “optimal” patterns, with respect to various conceptions of pattern quality.

The first abstraction was the CLOSEST SUBSTRING problem, which has been shown to have a polynomial time approximation scheme when restricted to a constant sized alphabet. In Section 4.1, a polynomial time approximation scheme was given for the problem with no assumptions about the size of the alphabet; the algorithm remains a PTAS even when $|\Sigma| \in \Omega(mn)$. The most restrictive assumption concerning alphabet size, the restriction to a binary alphabet, was shown to have a significantly greater approximability than previously known.

In Section 4.2 three different problems were analyzed, each with a different objective function. The MAX CLOSEST SUBSTRING problem, which is the complement of CLOSEST SUBSTRING, was shown NP-hard to approximate with performance ratio better than $\Omega(\log m)$. With a simple greedy algorithm, the problem was shown to be approximable within $|\Sigma| \log m$.

The objective of finding the largest subset $\mathcal{F}' \subseteq \mathcal{F}$ for which $\text{radius}(\mathcal{F}') \leq d$, for some specified d , was shown NP-hard to approximate with performance ratio better than $1 + 1/e$. No non-trivial upper bounds on the approximability of this problem are known.

For another objective, that of maximizing the length of a pattern, tight approximability bounds were obtained. The problem was abstracted as the LONGEST

Table 4.1: Approximability results for CLOSEST SUBSTRING problems with no restrictions place on $|\Sigma|$.

<i>Problem</i>	<i>NP-hard</i>	$\in P$
CLOSEST SUBSTRING	–	PTAS
MAX CLOSEST SUBSTRING	$\log m$	$ \Sigma \log m$
MAXIMUM COVERAGE APPROXIMATE SUBSTRING	$1 + 1/e$	–
LONGEST COMMON APPROXIMATE SUBSTRING	$2 - \epsilon$	2

COMMON APPROXIMATE SUBSTRING problem, which was shown NP-hard to approximate with performance ratio better than $2 - \epsilon$, for any $\epsilon > 0$. A 2-approximation algorithm was given for the problem, showing that the bound is tight. All of these results are summarized in Table 4.1.

There are many open questions concerning the approximability of the problems examined in this chapter. The complexity of CLOSEST SUBSTRING has been resolved, but it is possible that better approximation schemes exist for both CLOSEST STRING and CLOSEST SUBSTRING. In particular, it is likely that the occurrence set, found by random sampling in the LARGEALPHABET algorithm, can be obtained more efficiently.

Among the other objective functions analyzed in this chapter, the MAXIMUM COVERAGE APPROXIMATE SUBSTRING has a large margin for improvement in the complexity bounds. An efficient algorithm that gives even a constant time approximation, and even under the assumption of constant sized alphabet, would be valuable.

Chapter 5

Toward Optimal Enumeration

The COMMON APPROXIMATE SUBSTRING problem is a decision problem and therefore has a boolean solution set. It is difficult to conceive of situations where a boolean solution would be of use to biologists, except when the solution is negative (results for negative solutions, *i.e.* string *non-inclusion* problems, may be found in [77]). Practical uses for the COMMON APPROXIMATE SUBSTRING problem require the boolean solution to be accompanied by the center certifying the solution. The situation is similar for the optimization version, CLOSEST SUBSTRING. It is rarely the case that a practical problem can be solved by knowing that a set of strings has a particular *radius* unless the center achieving the radius is known. Results from Chapters 3 and 4 are useful for several reasons. First, lower bounds extend from the idealized problems to the more practical problems. More importantly, as is the case with any computational problem, analysis of the formal problems reveals mathematical properties that may be exploited in designing practical algorithms and heuristics.

A more appropriate formulation for the pattern discovery problem as it is used in practice is that of discovering *all* patterns that could be centers or closest substrings. In this chapter we will examine the problem under the name of MOTIF ENUMERATION, in an attempt to bring the problem abstraction closer to the reality of the motivating problems. As discussed in Chapter 2, the name *motif* is often used to refer to short patterns in bio-molecular sequences. The enumerative aspect corresponds to the fact that all motifs are equally important *a priori*. In the biological context, an advantage of the enumerative approach is that it allows the discovered motifs, which possess a certain combinatorial property, to be evaluated according to other criteria. In this capacity, the enumerative algorithms can provide input to other algorithms that filter motifs based on other properties.

We derive algorithms that produce the entire set of motifs satisfying the constraints of the problem parameters, and note that just as lower bounds on complexity for the COMMON APPROXIMATE SUBSTRING problem extend to MOTIF ENUMER-

ATION, all upper bounds on enumerating all motifs provide upper bounds on the corresponding decision problem. Formally, we define the MOTIF ENUMERATION problem as follows:

MOTIF ENUMERATION

Instance: The input is a set $\mathcal{F} = \{S_1, \dots, S_m\}$ of strings over an alphabet Σ such that $|S_i| \leq n$, $1 \leq i \leq m$, and integers l and d such that $0 \leq d < l \leq n$.

Question: The solution is a set of motifs $\mathcal{M}_{\mathcal{F}} \subseteq \Sigma^l$ such that for each motif $\mathcal{C} \in \mathcal{M}_{\mathcal{F}}$ and each $S_i \in \mathcal{F}$, there exists a length l substring of S_i that is Hamming distance $\leq d$ from \mathcal{C} .

The major computational challenges associated with enumerating motifs have already been demonstrated in Chapter 3, and to a lesser extent in Chapter 4. In addition to the difficulty presented by hardness of the underlying decision problem, we are faced with the task of producing all solutions. Fortunately, analysis of the FPT variants in Section 3.1 did not assume the algorithms halted after a solution was produced. So unlike the case of counting matchings in a graph, or satisfying DNF assignments, we do not expect a significant difference between the complexity of the decision and enumeration problems.

The chapter is organized as follows. In Section 5.1 we describe the first algorithm, CENSUS, that improves on an algorithm of Sagot [78] and establishes an upper bound on the time and space complexity that is linear in both the string length and the number of strings. We also discuss parallelizations of the algorithm. In Section 5.2, we describe the CIRCUITSIMULATION algorithm, further reducing the upper bound on the time complexity. The algorithm is based on a data structure that succinctly encodes sets of motifs, and allows efficient set operations. In Section 5.2.3, we show the problem admits an FPP algorithm, placing the corresponding decision version in the subclass of fixed parameter tractable problems that are highly parallelizable [18].

5.1 Improved Time and Space Complexity

There have been many enumerative algorithms for finding motifs in sets of sequences (see, *e.g.*, [92, 12, 88, 80, 86]). These algorithms each approach the problem differently; most attempt to eliminate as much of the search space as possible. Each, however, attempts to enumerate all strings of length l over the sequence al-

phabet. This most naive form of search introduces a factor of $\Omega(|\Sigma|^l)$ into the time complexity. The benefit of this type of enumeration is that it requires space bounded by a linear function of the size of the input. Further, when dealing with extremely large texts, the algorithm loses a small amount of efficiency when the text must be stored externally, but the motifs fit in memory.

Sagot [78] introduced a different approach that enumerates only those strings that are potential motifs, letting information from the sequences guide the enumeration. This more intelligent search remains within the (l, d) -neighborhood of each sequence. The method of Sagot has a time complexity of $O(lm^2nN)$, a space complexity of $O(lm^2n)$, and has proved successful in practice [89]. We note that the algorithm in [78] was actually designed with a “quorum” parameter, so that a motif is only required to be common to some $q \leq m$ of the sequences.

In this section we make an initial improvement to the known time and space complexity for the enumeration problem. We eliminate a factor of m from the requirements of the algorithm of Sagot [78]. This brings the time complexity to $O(lmnN)$, and the space complexity to $O(lmn)$. In Section 5.2 we further reduce the time complexity to $O(mnN)$.

5.1.1 The Census algorithm

The algorithm in [78] employed a generalized suffix tree [44], and required that each node indicate the subset of strings having the node’s label as a prefix. We eliminate the use of generalized suffix trees, and therefore eliminate the sets stored at each node. Analysis indicates that we have also eliminated the factor of $\Theta(m)$ in the time complexity without increasing the influence of the remaining parameters.

CENSUS begins with the construction, for each $S_i \in \mathcal{F}$, of the lexicographic trie T_i encoding all length l substrings of S_i , which requires $O(lmn)$ time. A lexicographic trie is a trie encoding a set of strings, where children of each node are ordered lexicographically (see the discussion in [1] for the similar *keyword tree*). The potential motifs of desired length l are not searched directly. The search process iteratively searches for each prefix of a given motif in order to take advantage of the fact that prefixes are shared by many potential motifs. This eliminates some redundant processing, as will be shown in the complexity analysis.

Let \mathcal{C} be a (length $\leq l$) motif for \mathcal{F} . Define the family of sets $F = \{F_1, \dots, F_m\}$ with respect to \mathcal{C} as

$$F_i = \{(v, k) : v \text{ is a node in the tree } T_i, \text{ and } 0 \leq k \leq d\},$$

where k counts mismatches between the label of v and \mathcal{C} , for each $1 \leq i \leq m$. Think of F_i as the frontier of nodes in T_i whose path labels are of Hamming distance $\leq d$ from \mathcal{C} . For any $(v, k) \in F_i$, the path label of node v spells out an occurrence of \mathcal{C} in S_i . For any $1 \leq i \leq m$ and element $(v, k) \in F_i$, the value of k is the number of mismatches between \mathcal{C} and the path label of node v in T_i . Given a character $\alpha \in \Sigma$ and a frontier set F defined with respect to a motif \mathcal{C} , the family of sets $F^\alpha = \{F_1^\alpha, \dots, F_m^\alpha\}$ is the frontier set defined with respect to the length $|\mathcal{C}| + 1$ motif $\mathcal{C}\alpha$.

While searching the space of possible motifs, if any $F_i \in F$ is found to be empty, the search space is pruned. The emptiness condition implies that there exists some member of \mathcal{F} containing no occurrence of the motif presently being searched. Pseudocode for the CENSUS algorithm is provided in Algorithm 9. For the initial call to CENSUS, \mathcal{C} is the empty string and F is the set of roots of the trees T_i , each given an error value of 0.

Algorithm 9: Pseudocode for the CENSUS algorithm.

Input: A set of lexicographic tries $F = \{F_1, \dots, F_m\}$ and a string \mathcal{C} .

Output: The set $\mathcal{M}_{\mathcal{F}}$ of motifs for \mathcal{F} .

CENSUS(\mathcal{C}, F)

1. **for each** character $\alpha \in \Sigma$
2. **for each** $F_i \in F$
3. **for each** $(v, k) \in F_i$
4. **if** node v has a child v' labeled with α
5. $F_i^\alpha \leftarrow F_i^\alpha \cup \{(v', k)\}$
6. **if** $k < d$
7. **for each** child v' of v that is not labeled with α
8. $F_i^\alpha \leftarrow F_i^\alpha \cup \{(v', k + 1)\}$
9. **if** $\forall F_i^\alpha \in F^\alpha, F_i^\alpha \neq \emptyset$
10. $\mathcal{C}' \leftarrow \mathcal{C}\alpha$
11. **if** $|\mathcal{C}'| = l$ **then output**(\mathcal{C}')
12. **else** make the recursive call CENSUS(\mathcal{C}', F^α)

The CENSUS algorithm was implemented in C and tested on a 1.4GHz Pentium 3 processor. Simulated data consisted of sequences generated uniformly at random from a 4 character alphabet. For (m, n, l, d) values of $(1000, 1000, 12, 3)$, CENSUS required 95 minutes and 370 megabytes of memory; 4.5 hours and 97 megabytes was required for values of $(100, 2000, 15, 4)$. The algorithm was also tested on a

dataset taken from the *E. coli* genome, with values (2645, 2000, 9, 2), and required 37 minutes and 991 megabytes of memory.

5.1.2 Complexity analysis

The space complexity of CENSUS is straightforward. Since the nodes of the lexicographic tries appear in at most one frontier at any instant, the space required is a constant multiple of the size of the lexicographic tries. We have the following bound on space complexity.

Theorem 5.1 *The space complexity of CENSUS is $O(lmn)$.*

The time complexity analysis is more intricate. We use an amortized analysis that bounds the total number of calls to CENSUS, and the average size of the frontier F passed to each call. We also analyze two modifications of the algorithm as it is described above. The first modification allows a quorum parameter, and the second changes the definition of the parameter d to represent an *average* distance instead of a maximum.

First we present a property of the search space of CENSUS, indicating the difficulty in eliminating the factor of $\Theta(l)$ that appears explicitly in the time and space complexity of the algorithm.

Lemma 5.1 *For any string $s \in \Sigma^l$, let $N_d(s) = \cup_{j=1}^l N_{j,d}(s[1..j])$, then*

$$d \in o(l/\log_{|\Sigma|}(l)) \Rightarrow |N_d(s)| \in \omega(N).$$

Proof. Consider the smallest lexicographic trie T encoding $N_d(s)$. Every simple path in T having the root as an endpoint corresponds to a string in $N_d(s)$. Equivalently, there is a bijection between $N_d(s)$ and the nodes of T . Each internal node in T must have degree $|\Sigma|$ children or 1 child, and no nodes with 1 child exist as ancestors of those having $|\Sigma|$ children. Let T' be portion of T consisting of nodes with $|\Sigma|$ children, along with their children. The number of nodes in T' is N , and the average depth of leaves in T' is

$$\lfloor \log_{|\Sigma|}(N) \rfloor < d(\log_{|\Sigma|}(l) + 1).$$

So the number of nodes in T having 1 child or no children is at least $N(l - d(\log_{|\Sigma|}(l) + 1))$. If $d \in o(l/(\log_{|\Sigma|}(s) + 1))$, then the total number of nodes in T is at least

$$N(l - d(\log_{|\Sigma|}(l) + 1)) \in lN(1 - o(1)) \in \omega(N).$$

□

The basic algorithm**Theorem 5.2** *The time complexity of CENSUS is $O(lmnN)$.*

Proof. The time complexity of the algorithm is proportional to the number of motifs in the search space, multiplied by the size of the family of frontiers F that must be constructed for each point in the search space. In the worst case, for m strings of length n , there are $O(nN)$ potential motifs for \mathcal{F} . The maximum size of the (l, d) -neighborhood of a string S of length n is $(n - l + 1)N$, and this is achieved when the d -neighborhoods of all length l substrings of S are disjoint. Further observe that this upper bound on the number of motifs in the search space gives an upper bound on the search space traversed by the algorithm when all (l, d) -neighborhoods of members of \mathcal{F} completely overlap. Attaining this limit on the search space requires that each $F_i \in F$ have exactly one element. For $1 \leq j \leq l$, let X_j be the space of all motifs \mathcal{C} for \mathcal{F} such that $|\mathcal{C}| = j$. Then under the condition of complete (l, d) -neighborhood overlap for members of \mathcal{F} :

$$|F| \sum_{j \leq l} |X_j| < mn \sum_{j \leq l} \binom{j}{d} (|\Sigma| - 1)^d = O(lmnN).$$

Consider how the search space is affected should any F_i have more than one element. The (l, d) -neighborhoods of substrings of S_i would no longer be disjoint and the d -neighborhood of S_i would have at least one fewer member. Since each increase in the size of a set $F_i \in F$, with respect to any motif \mathcal{C} , decreases the size of the search space by an equal amount, the situation of total (l, d) -neighborhood overlap is the worst case. Hence, the overall running time of the algorithm is $O(lmnN)$.

□

Including a quorum parameter

The algorithm can be easily adapted to handle the quorum version of the problem, as in [78]. Instead of requiring that each $F_i \in F$ be non-empty, it would be sufficient to verify that q of the frontiers be non-empty. In Section 4.2.2 the analogous optimization problem, MAXIMUM COVERAGE APPROXIMATE SUBSTRING, was examined.

Theorem 5.3 *Given a quorum parameter $q \leq m$, Census solves the motif discovery problem in $O((m/q)mnN)$ time and $O(mn)$ space.*

Proof. We must first bound the number of distinct motifs for a set of strings for a quorum value of q . If all strings are identical, the number of motifs is bounded by nN , regardless of q . As before, we consider the maximum number of possible motifs for each string, nN , and there are m strings. If the set of strings is partitioned into size m/q subsets, and the strings within each subset are identical, then each substring maps to N different motifs, and each motif has exactly q occurrences, which is optimal. \square

Modification for average distance

Another interesting related problem is obtained by modifying the relationship between a motif and its set of occurrences. For a motif \mathcal{C} , and a set K of occurrences of \mathcal{C} from distinct members of \mathcal{F} , define μ as

$$\mu = \min_{\mathcal{C} \in \Sigma^l} \sum_{S_i \in \mathcal{F}} \min_{1 \leq j \leq n-l+1} \left(\frac{d_H(s_{ij}, \mathcal{C})}{m} \right).$$

It is clear that $\mu \leq d$, now we are simply trying to minimize an average instead of a maximum. The modification simply requires maintaining the minimum number of mismatches between the center and a substring of S_i , for each $S_i \in \mathcal{F}$. This modification adds constant time for each *call* to the algorithm, the following complexity bound is obtained by bounding the number of calls. For the next theorem, we modify the definition of N , replacing the parameter d with μ .

Theorem 5.4 *For the modified problem where μ represents the average distance CENSUS can be modified to run in $O(|\Sigma|\mu lmnN)$ time and $O(|\Sigma|lmnN)$ space.*

Proof. We begin by imposing a score function over the set of strings in $N_{l,k}(s_{ij})$, ($0 \leq k \leq m\mu$), for any substring s_{ij} of any $S_i \in \mathcal{F}$. The score function χ is defined as follows:

$$\mathcal{C} \in N_{l,k}(s_{ij}) \Leftrightarrow \chi_{ij}(\mathcal{C}) = \mu - k + 1. \quad (5.1)$$

We can now characterize $\mathcal{M}_{\mathcal{F}}$ in terms of χ . For all $\mathcal{C} \in \mathcal{M}_{\mathcal{F}}$,

$$\mathcal{C} \in \mathcal{M}_{\mathcal{F}} \Leftrightarrow \forall S_i \in \mathcal{F}, \exists s_{ij} \in S_i, \sum_{i=1}^m \chi_{ij}(\mathcal{C}) \geq m. \quad (5.2)$$

To establish (5.2), we rephrase the definition of $\mathcal{M}_{\mathcal{F}}$ as

$$\mathcal{C} \in \mathcal{M}_{\mathcal{F}} \Leftrightarrow \forall S_i \in \mathcal{F} \exists s_{ij} \in S_i, \sum_{i=1}^m d_H(\mathcal{C}, s_{ij}) \leq m\mu. \quad (5.3)$$

That is, the sum of the differences between \mathcal{C} and each member of a set of occurrences, one from each string of \mathcal{F} , must be $\leq m\mu$ (this is by definition). Now, consider the sum of the χ_{ij} scores for \mathcal{C} :

$$\sum_{i=1}^m \chi_{ij}(\mathcal{C}) = \sum_{i=1}^m \mu - d_H(\mathcal{C}, s_{ij}) + 1 = m\mu - \left(\sum_{i=1}^m d_H(\mathcal{C}, s_{ij}) \right) + m. \quad (5.4)$$

Consider the equivalence expressed by 5.1. For the (\Rightarrow) direction, if $\mathcal{C} \in \mathcal{M}_{\mathcal{F}}$ then the right hand side of (5.3) holds. By substitution for the summation in (5.4), this implies that the summation must be $\geq m$. For the (\Leftarrow) direction, suppose the expression on the right hand side of (5.2) is true. By (5.4), this implies that the sum of distances is $\leq m\mu$, which by (5.3) implies that $\mathcal{C} \in \mathcal{M}_{\mathcal{F}}$.

We will bound the total number of motifs by optimally distributing the points of substrings of members of \mathcal{F} . The total number of points held by \mathcal{F} is

$$\begin{aligned} m(n-l+1) \sum_{k=0}^{\mu} (\mu - k + 1) \binom{l}{\mu} (|\Sigma| - 1)^{\mu} \\ \leq mn \sum_{k=0}^{\mu} \mu \binom{l}{k} (|\Sigma| - 1)^k = \mu mnN. \end{aligned} \quad (5.5)$$

Dividing this value by m , it follows that the maximum number of motifs for a set \mathcal{F} of strings is μnN .

Since the total size of all frontiers built during execution of the algorithm is $O(\mu lmnN)$, and each node in each frontier is examined $|\Sigma|$ times, the total complexity is $O(|\Sigma| \mu lmnN)$. \square

The bound on the number of motifs where μ is an average distance between motif and occurrence is unlikely to be tight. It seems reasonable to assume that the worst case is when all strings are identical.

Conjecture 5.1 *For the problem where μ represents the average distance between motif and motif occurrence, there can be no more than nN motifs for any set \mathcal{F} of strings, where N is defined with respect to μ .*

5.1.3 Parallelizations

The nature of the search in the algorithm makes it a prime candidate for distributed search. Practical aspects of such distributed searching are facilitated by the linear

space requirements (in terms of input size). The CENSUS algorithm can be parallelized to achieve $O(1)$ supersteps within the bulk-synchronous parallel (BSP) model [87]. BSP models parallelism using virtual processors that are mapped during execution to a smaller number of actual processors. An algorithm's computation is broken into *supersteps*, units of processing and communication that represent the necessary synchronization. All virtual processors must complete each specific superstep before any proceed to the next superstep. A parallel algorithm with a large superstep complexity is considered to be *fine grained*; it needs high synchronization and short time intervals. If the number of supersteps is small, the algorithm is *coarse grained* and requires little synchronization between virtual processors, which is desirable for parallel algorithms. The algorithm is modified as follows to partition the search space.

1. Each processor computes the prefixes of motifs for which it will search (these prefixes are distributed uniformly among the processors). Each processor searches for motifs, and when finished, broadcasts the number found.
2. With the information about the number of motifs each processor has found, processor P computes the starting address it will use to write the lexicographic trie into global memory. Then P writes the lexicographic trie encoding its motif set into global memory.

Theorem 5.5 *For all $1 \leq p \leq N$, the partitioned search space algorithm takes $O((N/p)mnl)$ time and requires $O(mnl)$ space per processor, while performing $O(1)$ supersteps.*

Proof. The time complexity of the supersteps follows from the time complexity of CENSUS. This algorithm only needs to be synchronized after each step in the modification description, so the number of supersteps is constant. \square

Another way to parallelize the algorithm is to assign each string in \mathcal{F} to a unique processor. Since the data for each input string is indexed in a separate lexicographic trie, this would be feasible for systems without shared memory. This type of parallelization reduces the influence of the number of sequences (m) on the time complexity of CENSUS.

1. Each processor constructs the lexicographic tries corresponding to its assigned sequences.

2. If the present motif prefix has length l , a motif has been found (a situation that can be handled in many ways). Otherwise, each processor makes the appropriate updates to the frontier sets based on the present motif prefix. When the frontiers have been updated, processors communicate whether or not to extend the present motif prefix, or backtrack. This step is repeated until the search is completed.

This is a fine grained parallelization, as the processors need to communicate after examining each extension. As such, we also consider the time for the communication required to direct the search.

Theorem 5.6 *Using p processors, the problem can be solved in $O(l(mn/p + \log p)N)$ time and $O(n)$ space per node.*

Proof. The factor of N in the time complexity of CENSUS corresponds to the search space that is traversed; this factor remains untouched by the parallelization. Similarly, the factor ln corresponds to the frontiers that must be maintained. Since disjoint sets of m/p frontiers are associated with distinct processors, they are updated independently in parallel, thus eliminating a factor of p from the time complexity. The only additional work to be accounted for is that required to determine when to backtrack. This requires communication, and essentially computing a logical “or” of a value from each processor. Done carefully, this requires $O(\log p)$ time. \square

5.2 Further Improvements

In this section we show how to solve the enumeration problem in $O(mnN)$ time, simultaneously giving an $O(mnN)$ upper bound on the time complexity of solving the COMMON APPROXIMATE SUBSTRING problem, which is presently the best known upper bound. The space complexity increases to $\Theta(mnN)$, but the output $\mathcal{M}_{\mathcal{F}}$ is produced in a concise encoding that requires $O(|\mathcal{M}_{\mathcal{F}}|)$ space, and can be queried for membership of a string s in time $O(|s|)$. The algorithm also has an efficient PRAM parallelization, which will be described in Section 5.2.3.

5.2.1 Simulating a boolean circuit

We begin by showing that the enumeration problem can be solved in $O(mnf(l, |\Sigma|))$ time and $O(mnf(l, |\Sigma|))$ space using sets of motifs and performing set operations.

We establish that log-depth computation, with respect to the input size, is sufficient for solving the problem. This is a computational expression of the fact that when N is a constant, the motif discovery problem can be expressed as a log depth boolean formula. The algorithm is an implementation of this logical characterization, which is similar to the characterization given in Section 3.3 by the *occurrence testing circuit*. The algorithm essentially replaces the disjunctions and conjunctions of the circuit with unions and intersections of motif sets. Our ultimate goal is a more efficient algorithm, but this algorithm demonstrates an important idea and motivates the details that will be addressed subsequently.

The algorithm is called CIRCUITSIMULATION, and is given in pseudocode in Algorithm 10. There is no special representation assumed for sets of motifs in the algorithm; the naive representation of a very large hash table is sufficient.

Algorithm 10: Computing $\mathcal{M}_{\mathcal{F}}$ using set operations

Input: A set \mathcal{F} of strings.

Output: The set $\mathcal{M}_{\mathcal{F}}$ of all motifs for \mathcal{F} .

CIRCUITSIMULATION(\mathcal{F})

1. $X \leftarrow \emptyset$
2. **for each** $S_i \in \mathcal{F}$
3. $X_i \leftarrow \emptyset$
4. **for each** length l substring s_{ij} of S_i
5. $X_{ij} \leftarrow N_{l,d}(s_{ij})$
6. $X_i \leftarrow X_i \cup X_{ij}$
7. **if** $X \neq \emptyset$ **then** $X \leftarrow X \cap X_i$
8. **else** $X \leftarrow X_i$
9. **return** X

It is easy to see that when $|\Sigma|^l \in O(1)$, the algorithm has a time complexity of $\Theta(mnf(l, |\Sigma|))$ time using $O(mnf(l, |\Sigma|))$ space. The time complexity is actually $\Theta(lmn|\Sigma|^l)$, and since $|\Sigma|^l \geq N$, we have not yet improved on the complexity bound given by CENSUS. We will reduce the complexity of the above algorithm by changing the way the sets are encoded, and providing more efficient algorithms for performing the set operations.

5.2.2 Neighborhood trees for set operations

We introduce a data structure that concisely encodes the (l, d) -neighborhood of a set of strings, while allowing some important operations to be performed efficiently. The data structure is similar in spirit to those from [47] and [6], which encode *subsequences* as small automata. Like motifs the subsequences are a type of approximation to a substring. Also like motifs, the number of subsequences of a string may be very large. Unfortunately our structure will be significantly larger than those for subsequences in order to support certain required operations efficiently: by not attempting to minimize the size of our structures we facilitate analysis of algorithms that operate on them.

The data structure is called a *neighborhood tree*, and is a compressed trie that uses indexing similar to that used by edge-compressed suffix trees.

Definition 5.2.1 (neighborhood tree) For any set \mathcal{F} of strings, each of length $n \geq l$, the (l, d) -neighborhood tree $T_{l,d}(\mathcal{F})$ for \mathcal{F} is a rooted directed tree satisfying four conditions:

1. Each edge is labeled with a string.
2. Any two edges out of the same node have labels beginning with distinct characters.
3. Any internal node has out-degree at least 2.
4. Every string $z \in N_{l,d}(\mathcal{F})$ maps to some leaf u of $T_{l,d}(\mathcal{F})$ such that the string formed by concatenating in order the labels on the path from the root to u exactly spell out z , and every leaf of $T_{l,d}(\mathcal{F})$ is mapped to some $z \in N_{l,d}(\mathcal{F})$.

When $\mathcal{F} = \{S\}$, we write $T_{l,d}(\mathcal{F})$ as $T_{l,d}(S)$ for convenience.

The (l, d) -neighborhood tree has the important property that, given a query string s of length l , it is capable of answering whether s is contained in the (l, d) -neighborhood of \mathcal{F} , and can do so in time proportional to size of the query (*i.e.* $|s|$). Our goal is to build and represent these structures in time and space proportional to the number of leaves in the tree, which is exactly $|N_{l,d}(\mathcal{F})|$. To accomplish this we must avoid explicitly representing the edge labels, as doing so would require $\omega(1)$ space per node. Our method is inspired by the edge compression used in linear time suffix tree algorithms [94, 62, 28], which represent edge labels by indexing substrings of

the underlying strings. The strategy does not transfer directly to neighborhood trees since not all substrings of members of $N_{l,d}(\mathcal{F})$ occur exactly as substrings of members of \mathcal{F} . The representation we use is based on an observation about $T_{l,d}(s)$ for a string s of length l .

Property 5.2.1 Let s be a string with $|s| = l$. For any edge $(u, v) \in T_{l,d}(s)$, if the string labeling edge (u, v) has length $x > 1$, then v is a leaf and the string labeling (u, v) differs by exactly 1 from the suffix of s having length x . In addition, the unique mismatch occurs at the first position of the string labeling edge (u, v) .

Thus, for the restricted case of an (l, d) -neighborhood tree for a string s with $|s| = l$, any edge label may be represented in constant space. It is sufficient to index the beginning and end of a substring in s , and indicate the character occupying the first position of the label. This special character is called the *modifier character*, and it will be assumed to exist on all edges, even those for which the modifier character is the same as the one indicated by the indexes into the underlying string.

We describe an algorithm to construct (l, d) -neighborhood trees for strings of length l . The reason for considering this restricted case is that the (l, d) -neighborhood tree for a string S of length $n > l$ can be obtained by taking the *union* of the (l, d) -neighborhood trees for each length l substring of S . The construction algorithm is based on the following recurrence. The symbol \circ denotes concatenation and when applied to a set operates on each member of the set.

Property 5.2.2 Let s and s' be strings with $|s| = l$ and $|s'| = l - 1$. If $s = \alpha \circ s'$ for some $\alpha \in \Sigma$, then

$$N_{l,d}(s) = \left(\alpha \circ N_{l-1,d}(s') \right) \cup \left(\cup_{\beta \in \Sigma} \beta \circ N_{l-1,d-1}(s') \right). \quad (5.6)$$

The recursive characterization of a neighborhood suggests a recursive algorithm for constructing a neighborhood tree. The information stored at each node in the tree includes numbers indexing a substring in the underlying string, and a modifier character that might override the first character indexed. We note that for this restricted case, the modifier alone is sufficient since edges with labels of length > 1 are incident on leaves, and their index is completely determined by the depth of the leaf and all leaves have the same depth. The reason for using the indexes is that they will be necessary later when constructing (l, d) -neighborhood trees for strings of length $> l$. We also anticipate the extension to neighborhood trees for sets of strings, and therefore assume the identity of each string is encoded along with the pair of indexes.

The construction algorithm is called BUILDNEIGHBORHOOD, and is given as pseudocode in Algorithm 11. The algorithm is essentially a computational realization of Property 5.2.2. The input is a single edge (the underlying string is assumed to always be available) connecting the root to the unique leaf. The two nodes are actually an $(l, 0)$ -neighborhood tree for some string s .

Algorithm 11: Pseudocode for the BUILDNEIGHBORHOOD algorithm.

Input: The root of an $(l, 0)$ -neighborhood tree for a string s . A distance parameter d .

Output: The root of an (l, d) -neighborhood tree $T_{l,d}(s)$ for s .

BUILDNEIGHBORHOOD(v, d)

1. Let x_α be the original child of v , and let α be the first character on the label of x_α .
2. **if** $d > 0$ and v is not a leaf
3. **for each** $\beta \in \Sigma \setminus \{\alpha\}$
4. Create child u_β of v with index $(\text{depth}(v) + 1, \text{depth}(v) + 1)$ and modifier character β .
5. Create child x_β of u_β with index $(\text{depth}(v) + 2, |s|)$ and no modifier character.
6. BUILDNEIGHBORHOOD($u_\beta, d - 1$)
7. Create node u_α with index $(\text{depth}(v) + 1, \text{depth}(v) + 1)$ and no modifier character. Increment the start index of x_α , insert x_α below u_α , and replace x_α with u_α as child of v .
8. BUILDNEIGHBORHOOD(u_α, d)
9. **return** v

Lemma 5.2 *For any string s , such that $|s| = l$, the (l, d) -neighborhood tree of s can be built in $O(N)$ time.*

Proof. First, notice that the total number of leaves in the tree is $O(N)$, since they are in 1-to-1 correspondence with members of $N_{l,d}(s)$. Because we use indexes instead of explicitly representing edge labels, the size of each node is constant, as is the time to create each node. Finally, since all nodes have out-degree ≥ 2 or 0 , the total number of nodes is proportional to the number of leaves in the tree. \square

After constructing $T_{l,d}(s_{ij})$ for each length l substring s_{ij} of each $S_i \in \mathcal{F}$, we take the union of these structures to obtain each $T_{l,d}(S_i)$. The intersection of all

$T_{l,d}(S_i)$ gives $T_{l,d}(\mathcal{F})$. For two (l, d) -neighborhood trees $T_{l,d}(s)$ and $T_{l,d}(s')$, corresponding to strings s and s' , the union $T_{l,d}(s) \cup T_{l,d}(s')$ is defined as the (l, d) -neighborhood tree encoding exactly the set of strings $N_{l,d}(s) \cup N_{l,d}(s')$. The intersection of neighborhood trees is defined similarly with respect to the intersection of the neighborhoods. The union and intersection operations are accomplished by recursively applying the operations to subtrees. This requires being able to determine the extent to which two nodes are identical, which requires determining the length of the longest identical prefix of a pair of edge labels. As an example, let $s = abcd$ and $s' = abba$, and consider $T_{4,0}(s)$ and $T_{4,0}(s')$ which each have two nodes. The union $T_{4,0}(s) \cup T_{4,0}(s')$ has four nodes (a root with one child that has two children), and three edge labels (ab , cd and ba). In order to determine the length (and label) of the edge labeled with ab while taking the union, we are required to determine the length of the longest prefix of s and s' . Our goal is to do this in constant time regardless of the length of the edge labels, so simply matching the strings does not suffice. The problem is handled using *longest common extension* queries, as explained in the proof of the next lemma.

Lemma 5.3 *The union and intersection operations for neighborhood trees can be performed in time bounded by a linear function of the size of the input structures.*

Proof. In a neighborhood tree resulting from a union or intersection operation, the number of nodes is bounded by a linear function of the total number of nodes in the trees being operated on. The union and intersection algorithms proceed by recursively applying unions and intersections on the appropriate subtrees of the input structures, and each node in the input structures need only be visited once. When the length of each edge label is $O(1)$, we need only spend constant time at each node in the input structures to determine the identity of the nodes to be created in the resulting structure. So for this restricted case, the set operations take linear time.

The only complication arises when two edge labels of length $\omega(1)$ must be compared to determine the length of their longest common prefix (as illustrated by the example above). Sequentially matching individual characters requires time proportional to the length of the shorter of the two edge labels. We use longest common extension queries to speed up the comparison. Given a pair of start indexes (i, j) for two substrings from (not necessarily distinct) strings S and S' , the longest common extension for (i, j) is the length of the longest prefix of suffix i of S that matches a prefix of suffix j of S' . The longest common extension for the starting indexes of two edge labels is equal to the length of the longest prefix that is identical in the

two labels.

After linear time preprocessing, longest common extension queries need only constant time. This is implemented by (1) creating a generalized suffix tree for the strings, which requires at most linear time using a number of methods, and (2) augmenting the tree for lowest common ancestor queries, which can also be done in linear time (for details see [44]). After creating and augmenting the generalized suffix tree, lowest common ancestor queries can be answered in constant time. The depth of the lowest common ancestor for two leaves gives the length of the longest common extension for the corresponding suffixes.

Therefore, even when arbitrary length edge labels are allowed, the edge labels can be compared in constant time during union and intersection operations. So linear time is sufficient for union and intersection operations in the general case. \square

While enumerative algorithms must have their running times dependent on the output size, we avoid this by encoding the set of motifs in a structure instead of producing each motif.

Theorem 5.7 *The time complexity of CIRCUITSIMULATION is $O(mnN)$.*

Proof. By Lemma 5.2, each call to BUILDNEIGHBORHOOD requires $O(N)$ time. There are $O(m)$ union operations, which by Lemma 5.3, each require $O(nN)$ time. Also by Lemma 5.3, the intersection operation requires at most $O(mnN)$ time. \square

Once the structure has been built, the complete list of motifs can be extracted in $O(lN)$ time, so the enumeration can be done in $O(mnN + lN) = O(mnN)$ time. The space requirements of the algorithm are the same as the time requirements.

5.2.3 A PRAM algorithm

Of more theoretical interest is the complexity of the problem when we are allowed a polynomial number of processors in the PRAM model of parallel computation, which has multiple processors using a shared memory. The class of algorithms that achieve a logarithmic running time using a polynomial number of processors is called NC [23]. Two analogues of NC have been defined within the context of parameterized complexity [18]. For a problem Π with parameter k , the class PNC

contains problems with algorithms requiring at most $O(f(k)(\log|x|)^{g(k)})$ time, using $O(h(k)|x|^c)$ processors, on instance $x \in \Pi$, for some constant c and arbitrary functions f, g, h . The definition of the class FPP modifies the allowed time to be $O(f(k)(\log|x|)^c)$, so $\text{FPP} \subseteq \text{PNC} \subseteq \text{FPT}$. The algorithm described in this section is a CREW-PRAM algorithm, meaning that the PRAM model is further qualified by the restriction that no two processors can write to the same memory location simultaneously.

The algorithm can be seen as an adaptation of the CIRCUITSIMULATION algorithm, where the set operations proceed from the leaves to the root of a binary processor tree. For any processor p at an internal node in this tree, subscripts L and R indicate data held by the left and right children of p . Leaf processors are assigned substrings of the strings in \mathcal{F} , and all processors assigned a substring from the same string are arranged consecutively. For the purpose of illustration, we assume both m and $n-l+1$ are powers of 2. Our algorithm requires that the neighborhood trees do not have compressed edges, so that construction of the suffix trees required for longest common extension queries may be avoided. The simple lexicographic tries used instead have a number of nodes bounded by $f(|\Sigma|, l) = O(|\Sigma|^l)$.

Algorithm 12: Pseudocode for the PARALLELCIRCUITSIMULATION algorithm.

Input: A set \mathcal{F} of strings.

Output: A lexicographic trie encoding $\mathcal{M}_{\mathcal{F}}$.

PARALLELCIRCUITSIMULATION(\mathcal{F})

1. **for each** leaf processor p (in parallel)
2. (p computes) $T_{ij} \leftarrow \text{BUILDNEIGHBORHOOD}(s_{ij})$
3. **for** $k \leftarrow 2$ **to** $\log(n-l+1)$
4. **for each** processor p (in parallel)
5. **if** p is at level k **then** (p computes) $T \leftarrow T_L \cup T_R$
6. **for** $k \leftarrow \log(n-l+1) + 1$ **to** $\log(m(n-l+1))$
7. **for each** processor p (in parallel)
8. **if** p is a level k **then** (p computes) $T \leftarrow T_L \cap T_R$
9. **return** T

The above algorithm establishes the following result concerning the parallel parameterized complexity of the MOTIF ENUMERATION problem.

Theorem 5.8 *The MOTIF ENUMERATION problem can be solved in $O(f(|\Sigma|, l) \log(mn))$ time using $O(mn)$ processors.*

5.3 Additional Applications

5.3.1 The Substring Parsimony problem

If the motif of interest is a biologically important pattern in a set of orthologous sequences (orthology describes sequences in different species that derive from a common ancestor), one might view the value d as representing some evolutionary distance between the motif and the set of occurrences. Extending this idea, one might refine the definition of d to reflect evolutionary distance connecting the motif to its set of occurrences. The l positions in the substring can be treated as characters, each having $|\Sigma|$ possible states. The SUBSTRING PARSIMONY problem, defined in [13], captures the idea of distance d being distributed over an evolutionary tree assumed to reflect evolutionary relationships between the sequences, and therefore the occurrences of the motif.

SUBSTRING PARSIMONY

Instance: The input is a family $\mathcal{F} = \{S_1, \dots, S_m\}$ of orthologous sequences from m different species, a phylogenetic tree T relating the species, and two integers l and d .

Question: The problem is to determine if there exist substrings s_1, \dots, s_m of S_1, \dots, S_m , respectively, such that the parsimony of s_1, \dots, s_m with respect to T is at most d .

Parsimony, in this context, is defined here as the sum of the distances between strings labeling adjacent nodes in the tree T . The principle of parsimony, in general, is an assumption in evolutionary theory that is similar to Occam's razor: the correct hypothesis is the one assuming the least.

Observe that in the above problem statement, there need not be a motif; the string at each node (leaves included) can be viewed as the motif. The problem can be modified placing a new node on any longest path in the tree such that the distance from the node to the ends of the path is $d/2$. The string at the new node would be analogous to the notion of a motif that we have been examining.

The problem was originally defined as an enumeration problem requiring the algorithm to produce all solutions. Unlike the MOTIF ENUMERATION problem, the number of solutions is not bounded by $|\Sigma|^l$ or nN . The problem specifies that a solution is a set of $\Theta(m)$ length l strings, so the combination is exponential in the number of leaves.

The complexity of this problem as demonstrated in [13] is given as $O(lmn(l(|\Sigma| -$

$1))^{d/2}$). We improve on this complexity here by combining techniques described in Section 5.2.1 and Section 5.2.3. We use the neighborhood tree to improve the complexity bound for this problem. We need the following property.

Lemma 5.4 *For any tree T and set of sequences \mathcal{F} , if the substring parsimony of \mathcal{F} with respect to T is d , then the optimal motif has distance at most $\lceil d/2 \rceil$.*

Proof. We assume d is even, the case of odd d is handled similarly. Fix some labeling of the nodes of T that realizes the parsimony. So every node is labeled with exactly one length l string. Consider a longest path P in T . If d is the sum of the lengths of the edges in T (i.e. the parsimony of \mathcal{F} with respect to T), then P has length at most d . Clearly, there are exactly 2 leaves on any longest path in T . Let v_1 and v_2 refer to those leaves, and let s_1 and s_2 be the strings labeling them, respectively. Let $\text{length}(P)$ be the sum of distances between strings labeling adjacent nodes on P . Let v_3 be the furthest node from v_1 in P that is at most $\text{length}(P)/2$ from v_1 . Let v_4 be the furthest node from v_2 in P that is at most $\text{length}(P)/2$ from v_2 , and not equal to v_1 . So if s_3 is the label for v_3 , and s_4 is the label for v_4 , then

$$d_H(s_1, s_3) + d_H(s_3, s_4) + d_H(s_4, s_2) = \text{length}(P).$$

Create a new node v_5 between v_3 and v_4 . Let s_5 be any string obtained by substituting $\text{length}(P)/2 - d_H(s_1, s_3)$ characters of s_4 for the characters of s_3 , and let s_5 label v_5 .

Now suppose s_5 is not a motif for \mathcal{F} with distance at most $d/2$. Then there is a string s labeling a leaf node such that $d_H(s, s_5) > d/2$. Without loss of generality, let $d_H(s, s_3) \geq d_H(s, s_4)$, so the shortest path from s to s_5 passes through s_3 . This implies

$$\begin{aligned} d_H(s, s_5) + d_H(s_5, s_2) &> d/2 + d_H(s_5, s_2) \\ &\geq d_H(s_1, s_5) + d_H(s_5, s_2) = \text{length}(P), \end{aligned} \quad (5.7)$$

contradicting the assumption that P is a longest path. \square

In the statement and proof of the next theorem, we assume N to be defined with respect to $d/2$, where d refers to the substring parsimony.

Theorem 5.9 *The SUBSTRING PARSIMONY problem can be solved in $O(mnN)$ time and $O(mnN)$ space.*

Proof. We describe the algorithm, which is based on the intersection of neighborhood trees, and the complexity will follow from the complexity of constructing and taking the intersection of neighborhood trees.

First, the $(l, d/2)$ -neighborhood tree for each $S_i \in \mathcal{F}$ is constructed, and each leaf stores the minimum distance between the string labeling that leaf and any substring of S_i . This modification does not alter the time complexity of the construction algorithm or the space required by the neighborhood trees. For each pair of sibling leaves, take the intersection of their trees, and let the tree held at the parent have d values equal to the sum of the values for the corresponding leaves in the children. Proceed in this manner until the next intersection is empty, or there are no remaining nodes. Continue this process in arbitrary order until (1) every node has a tree associated with it, or (2) the intersection of any set of sibling nodes is empty. When a single edge remains, take the intersection of the trees at the endpoints of the edge. If the intersection is empty, return negative, otherwise, the minimal d value for any motif in the intersection is the parsimony value. \square

5.3.2 The Distinguishing Substring problem

The motifs for a set of strings can also be looked upon as defining membership for that set. In biological applications such membership might also correspond to properties of hybridization or coregulation of associated genes. If the set of strings \mathcal{F} comes from a larger set, say \mathcal{U} , and membership is defined *a priori*, we might ask whether there exists a motif that can distinguish between \mathcal{F} and $\mathcal{F}' = \mathcal{U} \setminus \mathcal{F}$. This aspect, called the DISTINGUISHING SUBSTRING problem, was introduced in [54] under a slightly different definition.

DISTINGUISHING SUBSTRING

Instance: The input is two sets $\mathcal{F} = \{S_1, \dots, S_m\}$ and $\mathcal{F}' = \{S'_1, \dots, S'_m\}$, both over alphabet Σ , and three integers $d, d' < l$.

Question: The solution is a string $\mathcal{C} \in \Sigma^l$ such that (1) for each $S_i \in \mathcal{F}$, there exists a substring s_{ij} of S_i such that $d_H(s_{ij}, \mathcal{C}) \leq d$ and (2) for each $S'_i \in \mathcal{F}'$ and each substring s'_{ij} of S'_i , $d_H(s'_{ij}, \mathcal{C}) > d$.

We will assume symmetry with respect to the numbers of strings in the sets, the lengths of the strings, and the threshold distances. This is without loss of generality in our analysis, and we note that in any practical situation the values of these parameters are not likely to be the same.

Table 5.1: Time and space complexity of finding exact solutions.

<i>Reference</i>	<i>Space Complexity</i>	<i>Time Complexity</i>
Sagot [78]	$O(m^2n + lmn)$	$O(\Sigma lm^2nN)$
CENSUS	$O(lmn)$	$O(\Sigma lmnN)$
CIRCUITSIMULATION	$O(mnN)$	$O(mnN)$

A polynomial time approximation scheme was given for this problem in [25], and variants of this problem were shown hard for classes of the W-hierarchy in [41], where an algorithm was given for a restricted case. The next theorem establishes an upper bound on the time complexity of this problem that is equal to that of the motif discovery problem.

Theorem 5.10 *The DISTINGUISHING SUBSTRING problem can be solved in $O(mnN)$ time and $O(mnN)$ space.*

Proof. The algorithm constructs the neighborhood tree encoding $\mathcal{M}_{\mathcal{F}}$, which can be done in $O(mnN)$ time and space. Next, for each $S'_i \in \mathcal{F}'$, a neighborhood tree T_i is built in $m \times O(nN)$ total time. Finally, in m successive stages, the set difference $T \leftarrow T \setminus T'_i$ is performed. Each difference operation takes at most $O(nN)$ time. It follows that the total time required by the algorithm is $O(mnN)$. \square

5.4 Summary and Open Problems

Both sequential and parallel algorithms were derived for the MOTIF ENUMERATION problem. The sequential algorithms give new upper bounds on the complexity of solving both the enumeration problem, and the decision version examined in Chapter 3. The first parallel solutions to the problem have also been given. Notable from a theoretical perspective is the establishment of the COMMON APPROXIMATE SUBSTRING problem in the parallel parameterized complexity class FPP when $|\Sigma|$ and l are fixed. The sequential results are summarized in Table 5.1.

The major open problems deal with improving the space required to obtain a solution in $O(mnN)$ time. There is no evidence that the task is impossible, but there are significant challenges. Given the negative complexity results described in Chapter 3, it is unlikely that the time complexity can be greatly improved. Possible improvements include algorithms that require $O(mn + N)$ time.

The algorithm with the best worst case time bound is not expected to be practical, since the space requirements are $\Theta(mnN)$, and therefore so are the time requirements. If the space requirements were reduced by using minimal sized data structures in place of the neighborhood trees, the algorithm might yield efficient implementations.

Chapter 6

Conclusion

The research described in this thesis was motivated by pattern discovery problems arising in molecular biology. An abstraction was given in the form of the COMMON APPROXIMATE SUBSTRING problem. The complexity of this problem was examined from the perspective of both exact and approximate solutions. Algorithms and heuristics for the solution of COMMON APPROXIMATE SUBSTRING were also developed and tested.

This thesis has made several original contributions, both theoretical and practical. These contributions can be summarized as follows:

- The parameterized complexity of COMMON APPROXIMATE SUBSTRING has been systematically mapped. The large number of results produced a near complete complexity map, with only two parameterized variants left unresolved. On the positive side, it was shown that the non-trivial variant parameterized by input string length ($CAS(n)$) is fixed-parameter tractable. Hardness for parameterized complexity classes was shown with three distinct reductions (one of which was discovered independently in [32]), and membership was shown by the design of three distinct circuits.
- The first proof was given for the existence of a polynomial time approximation scheme for the CLOSEST SUBSTRING problem for input over an unrestricted alphabet. For the case of a binary alphabet, better upper bounds on existing algorithms were given; by the nature of approximation schemes, improved upper bounds imply more efficient algorithms. Results for hardness of approximation were obtained for optimization problems with four distinct objective functions. In spite of the approximation schemes previously developed for CLOSEST SUBSTRING, the negative results here indicate that many related optimization problems (having the same set of instances) differ significantly in terms of approximability.
- The algorithm CENSUS has the lowest time complexity of any known algo-

rithm for COMMON APPROXIMATE SUBSTRING. Reasons were given indicating why CENSUS might be optimal.

The research presented in this thesis has also uncovered some interesting questions for future research:

- What is the parameterized complexity of the remaining variants of COMMON APPROXIMATE SUBSTRING? In particular, can $CAS(|\Sigma|, d)$ be shown hard for any parameterized complexity class? The only known variants of COMMON APPROXIMATE SUBSTRING that are in FPT involve fixing l . Some surprising results due to Parida *et al.* [68, 69] concerning “maximal irredundant motifs” suggested that $CAS(m, |\Sigma|, d)$ might also be in FPT. Unfortunately, counterexamples were given in [72], indicating that the results were not valid.
- Better approximation algorithms certainly await discovery. What are the best ratios and corresponding time complexities? It seems reasonable to assume that the most important combinatorial properties of string sets have yet to be discovered; these could indicate much more efficient approximations.
- Can the CENSUS algorithm be improved to require only $O(mnN)$ time and space? What improvements, theoretical or practical, can be made by using smaller structures in the CENSUS algorithm. Can the algorithms based on neighborhood trees be made practical by using minimal DFAs or NFAs in their place?

Bibliography

- [1] A. Aho. Algorithms for finding patterns in strings. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 257–300. MIT Press / Elsevier, 1990.
- [2] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped blast and psi-blast- a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.
- [3] Stephen Altschul, Warren Gish, Webb Miller, Eugene Myers, and David Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.
- [4] Sanjeev Arora. *Probabilistic checking of proofs and the hardness of approximation problems*. PhD thesis, UC Berkeley, 1994.
- [5] Sanjeev Arora, David Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *Journal of Computer and System Sciences*, 58:284–293, 1995.
- [6] R. A. Baeza-Yates. Searching subsequences. *Theoretical Computer Science*, 78:363–376, 1991.
- [7] A. Bairoch. *The PROSITE database of protein families and domains user manual*. Swiss Institute of Bioinformatics, release 17 edition, December 2001.
- [8] Wayne M. Becker, Lewis J. Kleinsmith, and Jeff Hardin. *World of the Cell*, 4/e. Benjamin Cummings, 2000.
- [9] A. Ben-Dor, G. Lancia, J. Perone, and R. Ravi. Banishing bias from consensus sequences. In A. Apostolico and J. Hein, editors, *Proceedings of the*

- Annual Symposium on Combinatorial Pattern Matching*, number 1264 in Lecture Notes in Computer Science, pages 247–261, Aarhus, Denmark, 1997. Springer-Verlag, Berlin.
- [10] Otto G. Berg and Peter von Hippel. Selection of DNA binding sites by regulatory proteins II: The binding specificity of cyclic amp receptor protein to recognition sites. *Journal of Molecular Biology*, 200:709–723, 1988.
- [11] Edward A. Birge. *Bacterial and Bacteriophage Genetics, An Introduction*. Springer-Verlag, 1988.
- [12] M. Blanchette, B. Schwikowski, and M. Tompa. An exact algorithm to identify motifs in orthologous sequences from multiple species. In Bourne et al. [15], pages 37–45.
- [13] Mathieu Blanchette, Benno Schwikowski, and Martin Tompa. Algorithms for phylogenetic footprinting. *Journal of Computational Biology*, 9(2):211–223, 2002.
- [14] H. H. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett, and H. T. Wareham. Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences*, 11(1):49–57, 1995.
- [15] Philip Bourne, Michael Gribskov, Russ Altman, Nancy Jensen, Debra Hope, Thomas Lengauer, Julie Mitchell, Eric Scheeff, Chris Smith, Shawn Strande, and Helge Weissig, editors. *Proceedings of the Annual International Symposium on Intelligent Systems for Molecular Biology*, La Jolla, California, 2000. AAAI Press.
- [16] Alvis Brazma, Inge Jonassen, Ingvar Eidhammer, and David Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):277–304, 1998.
- [17] Peter J. Cameron. *Combinatorics: topics, techniques, algorithms*. Cambridge University Press, 1994.
- [18] Marco Cesati and Miriam Di Ianni. Parameterized parallel complexity. Technical Report 4(6), Electronic Colloquium on Computational Complexity (ECCC), 1997.

- [19] Marco Cesati and Luca Trevisan. On the efficiency of polynomial time approximation schemes. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(1), 1997.
- [20] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493 – 509, 1952.
- [21] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.
- [22] Stephen Cook. The P versus NP problem. MILLENNIUM PRIZE PROBLEMS, 2002.
- [23] Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1-3):2–21, 1985.
- [24] Francis Crick. Central dogma of molecular biology. *Nature*, 227:561–563, 1970.
- [25] Xiaotie Deng, Guojun Li, Zimao Li, Bin Ma, and Lusheng Wang. A ptas for distinguishing (sub)string selection. In *Proceedings of the Annual International Colloquium on Automata, Languages, and Programming*, pages 740–751, 2002.
- [26] R. Downey and M. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer-Verlag, New York, 1999.
- [27] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.
- [28] E.Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995.
- [29] P.A. Evans, A.D. Smith, and H.T. Wareham. The parameterized complexity of p -center approximate substring problems. Technical Report TR01-149, Faculty of Computer Science, University of New Brunswick, 2001.
- [30] P.A. Evans and H.T. Wareham. Practical algorithms for universal DNA primer design: An exercise in algorithm engineering (poster abstract). In N. El-Mabrouk, T. Lengauer, and D. Sankoff, editors, *Currents in Computational*

- Molecular Biology 2001*, pages 25–26, Montreal, PQ, 2001. Les Publications CRM.
- [31] Uriel Feige. A threshold of for approximating set cover. *Journal of the Association for Computing Machinery*, 45:634–652, 1998.
- [32] M.R. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of closest substring and related problems. In H. Alt and A. Ferreira, editors, *The 19th International Symposium on Theoretical Aspects of Computer Science STACS 2002*, volume 2285 of *Lecture Notes in Computer Science*, pages 262–273, Antibes/Juan-Les-Pins, France, 2002. Springer.
- [33] Alan Fersht. *Structure and mechanism in protein science: a guide to enzyme catalysis and protein folding*. W.H. Freeman and Company, 1999.
- [34] M. Frances and A. Litman. On covering problems of codes. *Theory of Computing Systems*, 30(2):113–119, 1997.
- [35] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981.
- [36] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Company; San Francisco, 1979.
- [37] Leszek Gasieniec, Jesper Jansson, and Andrzej Lingas. Efficient approximation algorithms for the hamming center problem. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.
- [38] Leszek Gasieniec, Jesper Jansson, and Andrzej Lingas. Approximation algorithms for hamming clustering problems. In *Proceedings of the Annual Symposium on Combinatorial Pattern Matching*, pages 108–118, 2000.
- [39] D. Gillman. A Chernoff bound for random walks on expander graphs. *SIAM Journal on Computing*, 27:1203–1220, 1998.
- [40] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete mathematics*. Addison-Wesley, Reading, MA, 1989.
- [41] J. Gramm, J. Guo, and R. Niedermeier. On exact and approximation algorithms for distinguishing substring selection. In *Proceedings of the 14th International Symposium on Fundamentals of Computation Theory (FCT 2003)*, 2003.

- [42] M. Gribskov, A. McLachlan, and D. Eisenberg. Profile analysis detection of distantly related proteins. *Proceedings of the National Academy of Sciences of the United States of America*, 88:4355–58, 1987.
- [43] T. Gura. Antisense has growing pains. *Science*, 270(27), 1995.
- [44] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
- [45] Dorit S. Hochbaum. *Approximation Algorithms for NP-hard Problems*, chapter 9: Good, Better, Best, and More, pages 346–398. PWS Publishing, 1996.
- [46] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.
- [47] W. Hsu and M. Du. Computing a longest common subsequence for a set of strings. *BIT*, 24:45–59, 1984.
- [48] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.
- [49] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.
- [50] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
- [51] J. Kececiloglu and R. Ravi. Of mice and men: Algorithms for evolutionary distances between genomes with translocation. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995.
- [52] P. G. Kolaitis and M. N. Thakur. Approximation properties of NP minimization classes. *Journal of Computer and System Sciences*, 50:391–411, 1995.
- [53] Anders Krogh, Michael Brown, I. Saira Mian, Kimmen Sjolander, and David Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.
- [54] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 633–642. ACM Press, 1999.

- [55] C. Lawrence and A. A. Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins: Structure, Function and Genetics*, 7:41–51, 1990.
- [56] H. W. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4), 1983.
- [57] M. Li, B. Ma, and L. Wang. Finding similar regions in many sequences. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 473–482. ACM Press, 1999.
- [58] Ming Li, Bin Ma, and Lusheng Wang. Finding similar regions in many strings. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 473–482, 1999.
- [59] Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *Journal of the ACM*, 49(2):157–171, 2002.
- [60] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the Association for Computing Machinery*, 41(5), 1994.
- [61] Bin Ma. A polynomial time approximation scheme for the closest substring problem. In R. Giancarlo and D. Sankoff, editors, *Combinatorial Pattern Matching (CPM 2000)*, volume 1848 of *Lecture Notes in Computer Science*, pages 99–107. Springer-Verlag, 2000.
- [62] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the Association for Computing Machinery*, 23(2):262–272, 1976.
- [63] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [64] Kary B. Mullis. The unusual origin of the polymerase chain reaction. *Scientific American*, 262:56–65, 1990.
- [65] James A.H. Murray, editor. *Antisense RNA and DNA*. Wiley-Liss, Inc., 1993.
- [66] Donald W. Nicholson. From bench to clinic with apoptosis-based therapeutic agents. *Nature*, 407:810–816, 2002.

- [67] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.
- [68] Laxmi Parida. *Algorithmic Techniques in Computational Genomics*. PhD thesis, Courant Institute of Mathematical Sciences, New York University, September 1998.
- [69] Laxmi Parida, Isidore Rigoutsos, Aris Floratos, Dan Platt, and Yuan Gao. Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and an efficient polynomial time algorithm. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 297–308, 2000.
- [70] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States of America*, 85:2444–2448, 1988.
- [71] P. Pevzner and S. Sze. Combinatorial approaches to finding subtle signals in DNA sequences. In Bourne et al., editor, *Proceedings of the Annual International Symposium on Intelligent Systems for Molecular Biology*, pages 269–278. AAAI Press, 2000.
- [72] Nadia Pisanti, Maxime Crochemore, Roberto Grossi, and Marie-France Sagot. A basis of tiling motifs for generating repeated patterns and its complexity for higher quorum. In B. Rován and P. Vojtáš, editors, *MFCS, LNCS*. Springer-Verlag, 2003. To appear. See TR-03-02, Università di Pisa.
- [73] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.
- [74] Prabhakar Raghavan and Clark D. Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.
- [75] Prabhakar Raghvan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988.

- [76] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 475–484, 1997.
- [77] Anatoly R. Rubinov and Vadim G. Timkovsky. String noninclusion optimization problems. *SIAM Journal on Discrete Mathematics*, 11(3):456–467, 1998.
- [78] Marie-France Sagot. Spelling approximate repeated or common motifs using a suffix tree. In C. L. Lucchesi and A. V. Moura, editors, *Proceedings of the Third Latin American Symposium on Theoretical Informatics*, volume 1390 of *Lecture Notes in Computer Science*, pages 374–390. Springer, 1998.
- [79] Sartaj Sahni. General techniques for combinatorial approximation. *Operations Research*, 25(6):920–936, 1977.
- [80] S. Sinha and M. Tompa. A statistical method for finding transcription factor binding sites. In Bourne et al. [15], pages 344–344.
- [81] John Maynard Smith. *Evolutionary Genetics*. Oxford University Press, 1989.
- [82] Rodger Staden. Methods for calculating the probabilities of finding patterns in sequences. *Computer Applications in the Biosciences*, 5(2):89–96, 1989.
- [83] G. Stormo and G.W. Hartzell III. Identifying protein-binding sites from unaligned DNA fragments. *Proceedings of the National Academy of Sciences of the United States of America*, 86:1183–1187, 1989.
- [84] Tom Strachan and Andrew P. Read. *Human Molecular Genetics*, 2/e. John Wiley & Sons Inc., 1999.
- [85] L. Stryer. *Biochemistry*. W. H. Freeman, New York, 4 edition, 1994.
- [86] Martin Tompa. An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. In *Proceedings of the Seventh Proceedings of the Annual International Symposium on Intelligent Systems for Molecular Biology*, 1999.
- [87] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

- [88] 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.
- [89] A. Vanet, L. Marsan, A. Labigne, and M.-F. Sagot. Inferring regulatory elements from a whole genome. an analysis of the *helicobacter pylori* σ^{80} family of promoter signals. *Journal of Molecular Biology*, 297:335–353, 2000.
- [90] H.T. Wareham. *Systematic Parameterized Complexity Analysis in Computational Phonology*. PhD thesis, Department of Computer Science, University of Victoria, 1999.
- [91] H.T. Wareham. Personal Communication, 2001.
- [92] M. S. Waterman, R. Arratia, and D. J. Galas. Pattern recognition in several sequences: consensus and alignment. *Bulletin of Mathematical Biology*, 46:515–527, 1984.
- [93] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171:737–738, 1953.
- [94] Peter Weiner. Linear pattern matching algorithms. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 1–11, 1973.
- [95] C. T. Workman and G. D. Stormo. Ann-spec: a method for discovering transcription factor binding sites with improved specificity. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 467–478, 2000.

Appendix A

Circuit Diagrams

This appendix gives diagrams of the circuits described in Section 3.3. At each level of the circuits is an explanation of the function of the gates at that level. Recall that for complexity purposes, depth is only required to be independent of input size for the lower classes of the W -hierarchy (*i.e.* $W[t]$ for constant t); weight is what we would like to minimize. The number of inputs is restricted to be a polynomial function of the problem size, but the weight of any assignment to those inputs must be a function of the parameters. The part of each circuit enforcing that any satisfying assignment correspond to a valid occurrence has is not described. The symbol \vee represents the inclusive *OR* function and the symbol \wedge represents *AND*. The fan-in of each gate is given, and may or may not be restricted depending on which parameters are fixed.

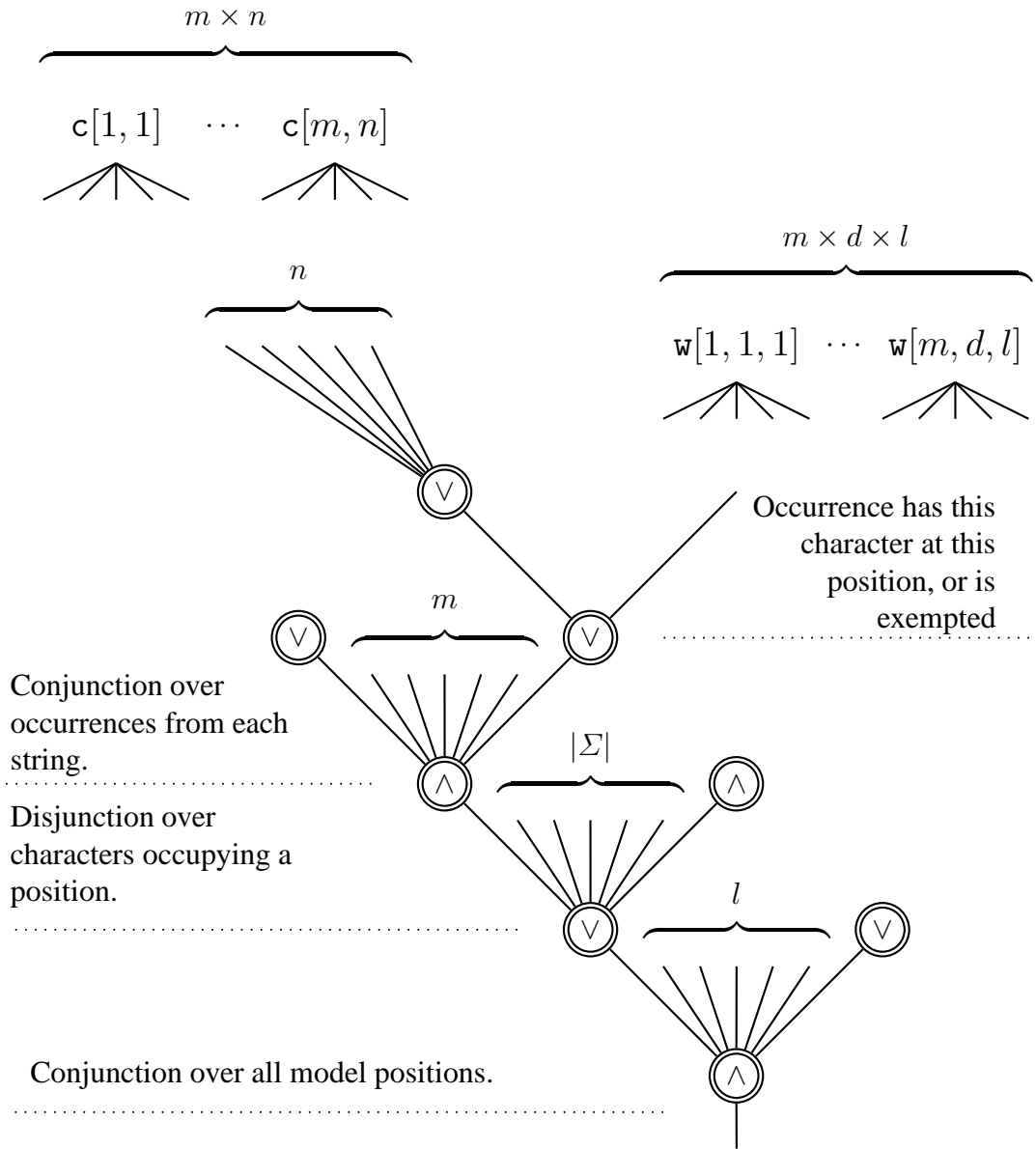


Figure A.1: Configuration of the instance testing circuit.

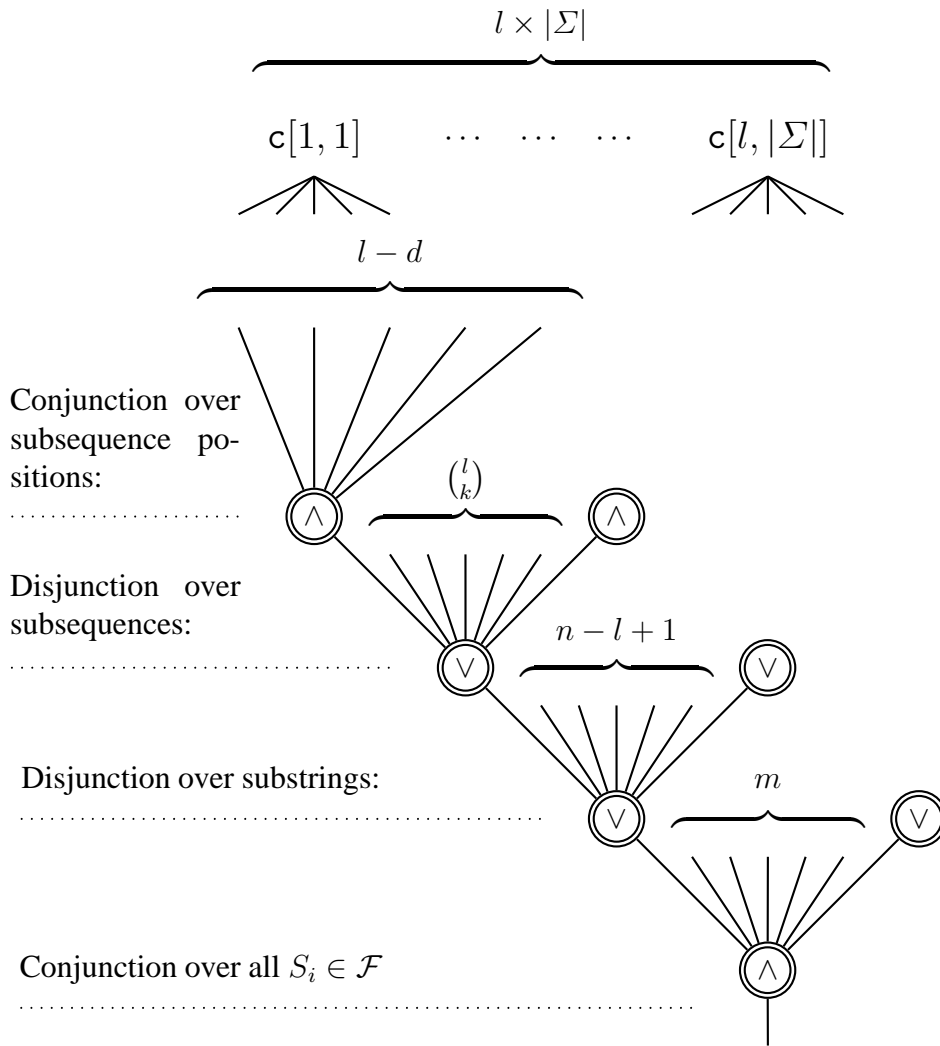


Figure A.2: Configuration of the center testing circuit.

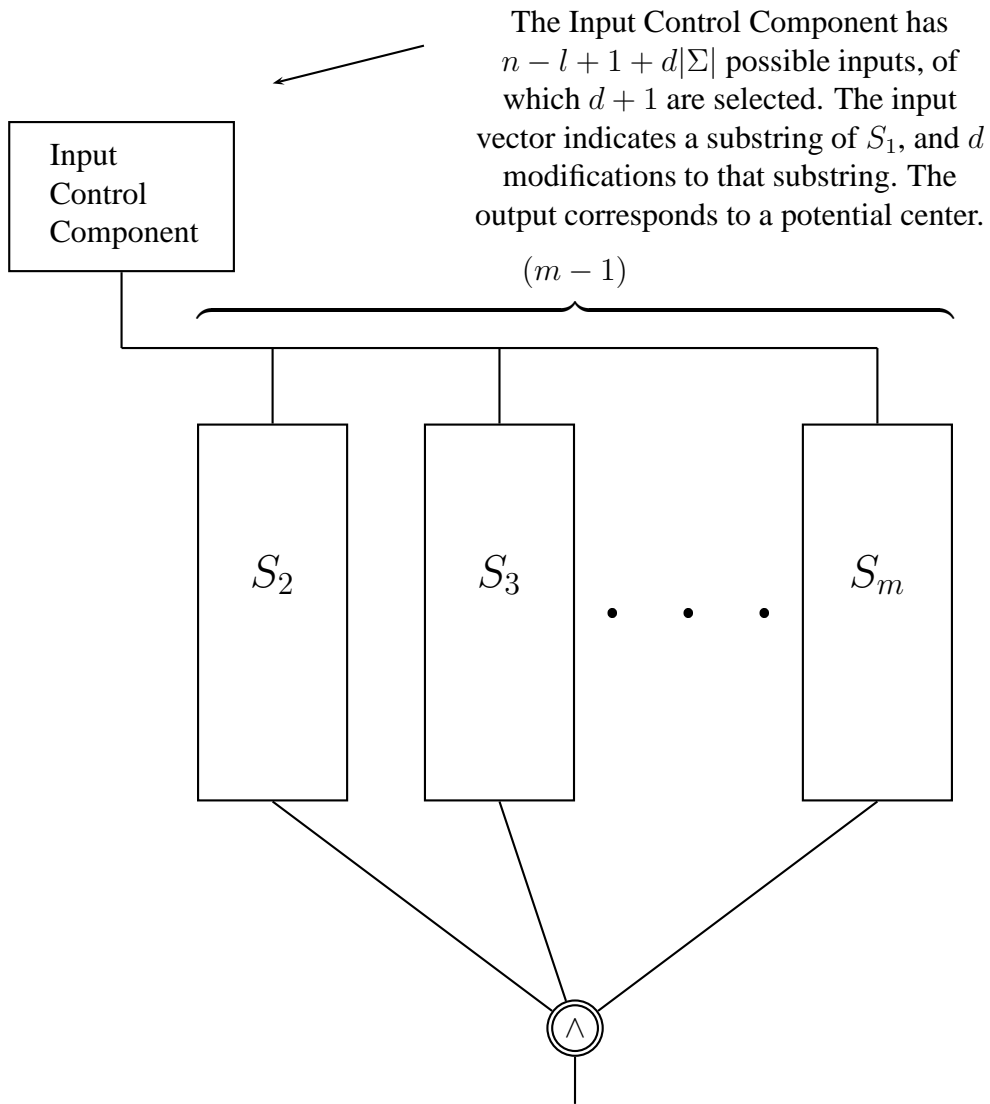


Figure A.3: High level configuration of the single occurrence + modifications testing circuit.

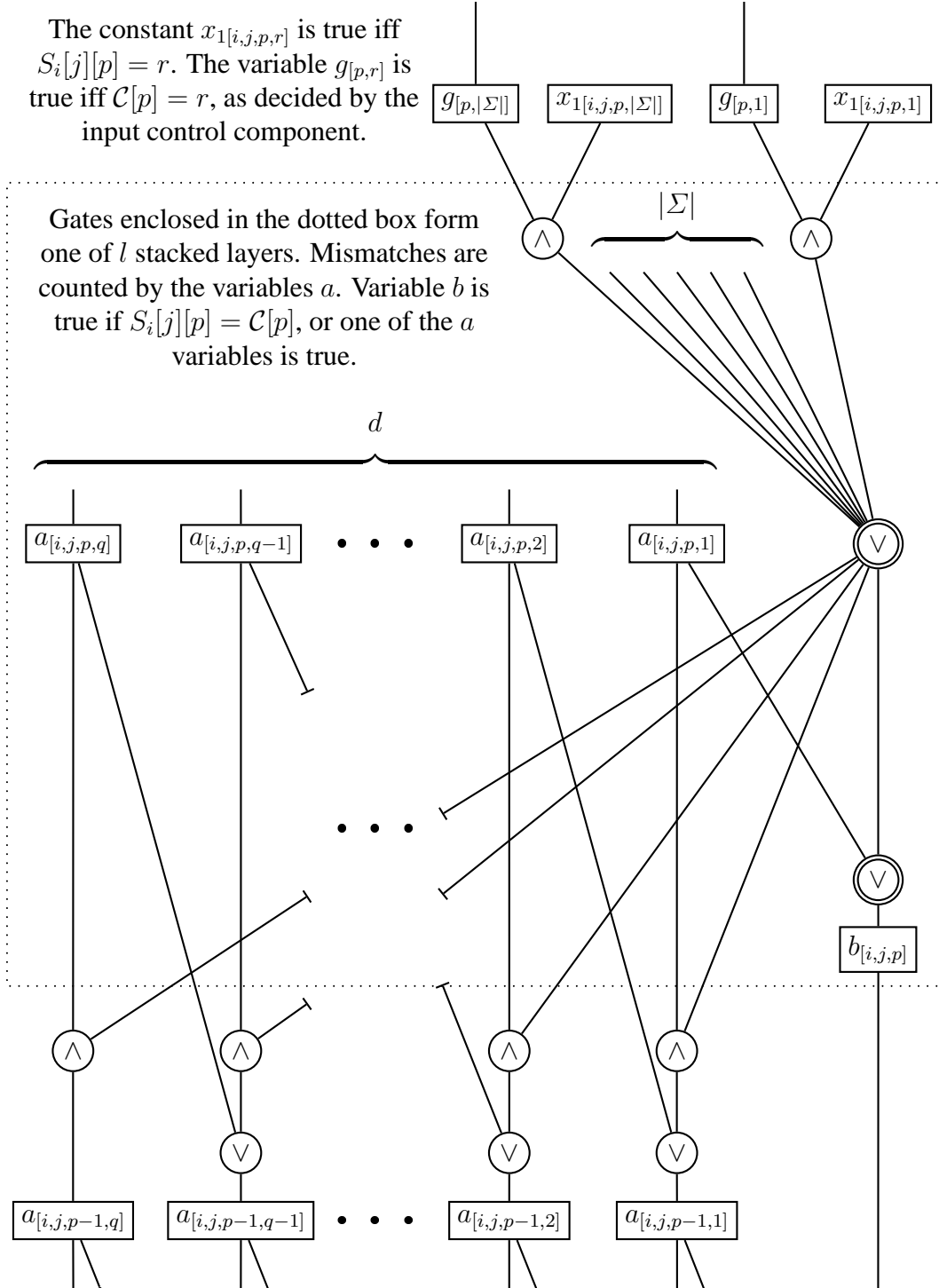


Figure A.4: Mismatch counting part of the single instance + modifications testing circuit.

This part of the circuit appears $l \times |\Sigma|$ times, once for each possible modification to the occurrence selected from S_1 . The inputs to the left (x_2 variables) indicate the selection of a substring from S_1 . The $x_3[p, r]$ variable indicates whether the center is modified to contain character r at position p . The negated inputs correspond to all other characters. Although not indicated in the diagram, $x_3[p, r]$ does not appear negated with the other x_3 's. The only appearance of $x_3[p, r]$ is directly above the OR gate feeding into the variable $g[p, r]$.

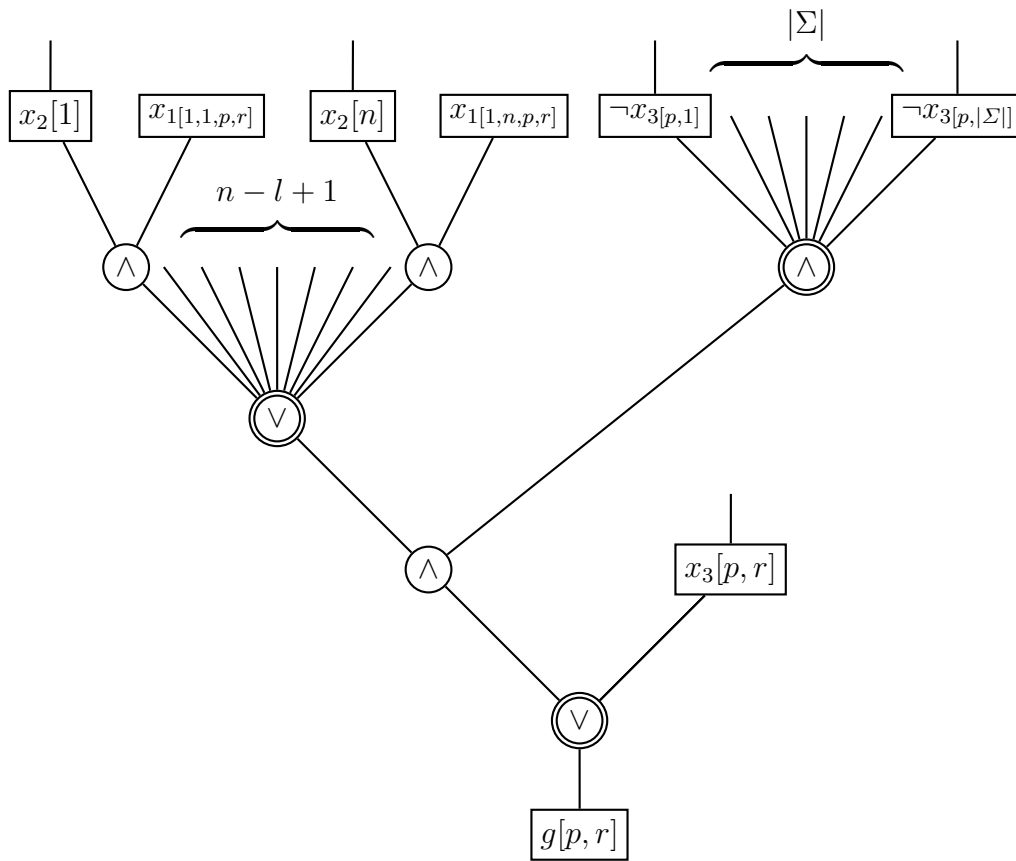


Figure A.5: Inputs to the single occurrence + modifications testing circuit.

VITA

Andrew David Smith

- February 28, 1977 Born at Chatham, New Brunswick, Canada
- 1995-2000 B.A.(Psychology), University of New Brunswick, Fredericton
- 1997-2000 B.C.S., University of New Brunswick, Fredericton

PUBLICATIONS

- Patricia A. Evans, Andrew D. Smith, and H.T. Wareham. The parameterized complexity of p -center approximate substring problems. Technical Report TR01-149, Faculty of Computer Science, University of New Brunswick, 2001.
- Patricia A. Evans and Andrew D. Smith. Complexity of approximating closest substring problems. In Andrzej Lingas and Bengt J. Nilsson, editors, *Proceedings of the 14th International Symposium on Fundamentals of Computation Theory (FCT 2003)*, volume 2751 of *Lecture Notes in Computer Science*, pages 210–221. Springer-Verlag, 2003.
- Patricia A. Evans and Andrew D. Smith. Toward optimal motif enumeration. In Frank Dehne, Jorg-Rudiger Sack, and Michiel Smid, editors, *Proceedings of the 8th International Workshop on Algorithms and Data Structures (WADS 2003)*, volume 2748 of *Lecture Notes in Computer Science*, pages 47–58. Springer-Verlag, 2003.
- Patricia A. Evans, Andrew D. Smith, and H. T. Wareham. On the complexity of finding common approximate substrings. *Theoretical Computer Science*, 306(1-2):407–430, 2003.

Andrew D. Smith. A W[2]-hard variant of common approximate substring. Technical Report TR02-156, Faculty of Computer Science, University of New Brunswick, 2002.

Andrew D. Smith, Thomas W. H. Lui, and Elisabeth R. M. Tillier. An empirical model for substitution in ribosomal RNA. *Molecular Biology and Evolution*, 2003. In Press.

S. Veerassamy, Andrew D. Smith, and Elisabeth R. M. Tillier. A transition probability model for amino acid substitutions from blocks. *Journal of Computational Biology*, 2003. In Press.

FIELD OF STUDY

Computer Science (Algorithms, Computational Biology, Computational Complexity)

Common Approximate Substrings

ABSTRACT

Discovering patterns in strings is a central task in analyzing molecular sequences. One pattern discovery problem is to find a pattern that occurs as a substring in each member of a given set of strings. Additionally, occurrences of this pattern are allowed to have up to some specified number of errors, so the occurrences may not exactly match the pattern. Allowing such “approximate occurrences” significantly complicates methods for discovering the pattern, rendering it NP-hard.

The pattern discovery problem is abstracted as a decision problem under the name Common Approximate Substring. A systematic parameterized complexity analysis is conducted, producing a nearly complete parameterized complexity map with respect to the number of input sequences, their maximum length, the length of the pattern, the maximum number of mismatches between the pattern and its occurrences, and the size of the sequence alphabet. The analysis has also revealed several new results, including the first FPT variant not parameterized with alphabet size.

The Closest Substring problem is an optimization problem with the goal of minimizing the maximum number of mismatches between the pattern and any occurrence. Previous studies did not resolve the effect of alphabet size on approximability. The problem admits a polynomial time approximation scheme even for the case of an unrestricted alphabet, and an improvement is given for the case of a binary alphabet. Objective functions are derived from three other aspects of the problem, and new approximability results are described for each associated optimization problem. The results include both upper and lower bounds, and in one case these bounds are shown to be tight.

In practice one might only know some set of necessary conditions for important patterns, and desire the identity of each pattern matching those conditions. This goal is captured by the problem of enumerating all patterns fitting a specified length and number of allowed errors. New upper bounds are proved for the complexity of the enumeration problem, and parallelizations are described that have both practical and theoretical value. The upper bound on the complexity is achieved through the use of a new data structure capable of concisely encoding sets of patterns, while providing efficient membership queries and efficient set operations.

Andrew David Smith
Faculty of Computer Science
University of New Brunswick
Advisor: Patricia A. Evans
Common Approximate Substrings