The Parameterized Complexity of p-Center Approximate Substring Problems

Patricia A. Evans[1], Andrew D. Smith[1], and H. Todd Wareham[2]

Abstract

Problems associated with finding strings that are within a specified Hamming distance of a given set of strings occur in several disciplines. All of the problems investigated are NP-hard and have varying levels of approximability. In this paper, we use techniques from parameterized computational complexity to assess non-polynomial time algorithmic options for three of these problems, namely p-**EXACT SUBSTRING** (p**ES**), **APPROXIMATE SUBSTRING** (1**AS**), and p-**APPROXIMATE SUBSTRING** (p**AS**). These problems vary whether the substring must be an exact match, and also whether a single substring or a set of substrings (of cardinality p) is required. Our analyses indicate under which parameter restrictions useful algorithms are possible, and include both class membership and parameterized reductions to prove class hardness. Since variation in parameter restrictions will lead to different algorithms being preferable, we give a variety of algorithms for the fixed parameter tractable problem variations. One of these, for 1**AS** with alphabet, substring length, and distance all fixed, is an improvement of one of the best previously known exact algorithms (under these restrictions). Other algorithms solve parameterized variants previously unexplored. We also prove that p**ES** is NP-hard, and show inapproximability for p**ES** and p**AS**.

[1]Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada. E-mail: {pevans,p7ka}@unb.ca

[2]Department of Computer Science, Memorial University of Newfoundland, St. John's, NF, Canada. E-mail: harold@cs.mun.ca

1 Introduction

Given two strings x and y of the same length over an alphabet Σ, the **Hamming distance** between x and y is the number of positions at which the symbols in x and y differ. Over the last several years, a number of results have been derived for restricted versions of the following problem involving Hamming distance:

p-APPROXIMATE SUBSTRING (pAS)

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

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

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

When $l = n$ and $p = 1$, pAS is known as **COVERING RADIUS** [6] and has been investigated in the context of coding theory. When $l = n$ and $p \geq 1$, pAS is known as **HAMMING RADIUS p-CLUSTERING** [8] and has been investigated in the context of string clustering. When $l \leq n$ and $p = 1$, pAS is known as **CLOSEST SUBSTRING** [12, 13] and has been investigated in the context of DNA probe design in molecular biology. All of these variants have been shown to be NP-hard. All of these aspects can be seen as existing in a space relative to pAS defined by three dimensions:

1. Length of requested string relative to length of given strings ($l \leq n$ vs. $l = n$);
2. Number of requested substring "centers" ($p = 1$ vs. $p \geq 1$); and
3. Exactness of match of requested and given strings ($d = 0$ vs. $d \geq 0$).

Though several members of this space are admittedly trivial (pAS when $l = n$, $p \geq 1$, and $d = 0$) or are known to be solvable in low-order polynomial time (pAS when $l \leq n$, $p = 1$, and $d = 0$, *i.e.*, **LONGEST COMMON SUBSTRING**), the NP-hardness of the problems mentioned above suggest that the remainder of the problems in this space may be difficult. A patchwork of associated polynomial-time approximation algorithms and inapproximability further hints at interesting relationships as well as differences between these problems.

In this paper, we will provide a unified treatment of three versions of pAS existing along the second and third dimensions listed above when $l \leq n$ – namely, pAS when $p \geq 1$ and $d \geq 0$, **p-EXACT SUBSTRING** (pES; pAS when $p \geq 1$ and $d = 0$), and **APPROXIMATE SUBSTRING** (1AS; pAS when $p = 1$ and $d \geq 0$). In particular, we will use techniques developed within the theory of parameterized computational complexity [4] to systematically examine

the possible types of useful non-polynomial time algorithms for these problems. Such systematic treatments are useful both in selecting algorithms that will operate most efficiently on instances of these problems that occur in practice and in guiding research on new algorithms for these problems [23]. In general, several parameters must be restricted together in order to produce a useful algorithm. For 1**AS**, the substring length l needs to be restricted together with the alphabet size $|\Sigma|$, and more effective algorithms result if additional parameters are also restricted. For p**ES**, it is sufficient to restrict the number of strings m in order to obtain fixed parameter tractability. Further results give class membership and hardness for many other parameter variations of the problems. We also prove that p**ES** is NP-hard, show polynomial-time inapproximability for p**ES** and p**AS**, and give a new algorithm for 1**AS** that improves on one of the best previously known exact algorithms [19].

Parameterized complexity analysis: According to the theory of NP-completeness [7], an NP-hard problem does not have a polynomial time algorithm (and hence cannot be solved quickly for all instances) modulo the strength of the conjecture that $P \neq NP$. However, restricted instances of some NP-hard problems encountered in practice can be solved quickly by invoking non-polynomial time algorithms. This is because the non-polynomial terms in the running times of these algorithms are purely functions of sets of aspects of the problems that are of bounded size or value in those instances, where an **aspect** of a problem is some (usually numerical) characteristic that can be derived from instances of that problem, *i.e.*, $|\Sigma|$, d, and l in the case of p**AS**. The theory of parameterized computational complexity [4] provides explicit mechanisms for analyzing the effects of individual aspects on problem complexity. Given a decision problem Π with a parameter p, let $\Pi(p)$ denote the parameterized problem associated with Π that is based on parameter p. To allow us to isolate components of problem descriptions and hence parameters, we broadly define an **aspect** of a problem as a characteristic that can be derived from instances of that problem, *i.e.*, $|\Sigma|$, d, and l in the case of p**AS**.

Definition 1.1 *A parameterized problem $\Pi(p)$ is* **fixed parameter tractable** *if there exists an algorithm A to determine if instance x is in $\Pi(p)$ in time $f(p) \cdot |x|^{\alpha}$, where $f : \Sigma^{+} \mapsto \mathcal{N}$ is an arbitrary function and α is a constant independent of x and p.*

One can establish that a parameterized problem Π is not fixed parameter tractable by using a parametric reduction[3] to show that $\Pi(p)$ is hard for any of the classes of the **W-hierarchy** $= \{FPT, W[1], W[2], \ldots, W[P], XP\}$ except FPT, where FPT is the class of fixed parameter tractable parameterized problems (see [4] for details). These classes are related as follows:

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

[3]Given parameterized problems $\Pi(p)$ and $\Pi'(p')$, $\Pi(p)$ **parametrically reduces to** $\Pi'(p')$ if there is an algorithm A that transforms an instance x of $\Pi(p)$ into an instance x' of $\Pi'(p')$ such that A runs in time $f(p)|x|^{\alpha}$ time for an arbitrary function f and a constant α independent of both x and p, $p' = g(p)$ for some arbitrary function g, and $x \in \Pi(p)$ if and only if $x' \in \Pi'(p')$.

If a parameterized problem can be shown to be $\mathcal{C}$-hard for any class $\mathcal{C}$ in the W hierarchy above FPT, then that problem is not in FPT (and hence is not fixed parameter tractable) modulo the strength of the conjecture that $FPT \neq \mathcal{C}$.

The following lemmas will be used in the analyses given in later sections of this paper.

Lemma 1.1 [16, Lemma 2.1.25] *Given decision problems* Π *and* Π' *with parameters* p *and* p'*, respectively, if* $\Pi \leq_m \Pi'$ *such that* $p' = g(p)$ *for an arbitrary function* g*, then* $\Pi(p)$ *parametrically reduces to* $\Pi'(p')$.

Lemma 1.2 [16, Lemma 2.1.35] *Given a set* S *of aspects of a decision problem* Π*, if* Π *is* NP*-hard when the value of every aspect* $s \in S$ *is fixed, then the parameterized problem* $\Pi(S)$ *is not in* XP *unless* $P = NP$.

Note that in the lemmas above and the remainder of this paper, we use the symbol "$\leq_m$" to denote a polynomial-time many-one reduction between decision problems.

Organization of this paper: Sections 2, 3 and 4 give our results for the 1**AS**, p**ES** and p**AS** problems, respectively. There are two parts to each section; the first describes efficient algorithmic solutions relative to certain problem aspects, the other thoroughly analyzes the parameterized complexity relative to aspects for which the problems are not known to be fixed parameter tractable. All parameterized analysis in these sections were done relative to the following aspects: the size of the alphabet ($|\Sigma|$), the number of strings in $\mathcal{F}$ (m), the length of the strings in $\mathcal{F}$ (n), the number of "centers" (p), the length of the requested substrings (l), and the Hamming distance threshold (d). While n is usually considered to describe the input size of an instance, we analyse it as a parameter for the purpose of completeness. Section 5 lists some promising directions for future research.

2 APPROXIMATE SUBSTRING

This problem is also known as **CLOSEST SUBSTRING** [12, 13]. It is NP-hard by results derived independently in [6] and [12] for **CLOSEST STRING**, *i.e.*, **APPROXIMATE SUBSTRING** when $l = n$. Several polynomial-time approximation algorithms that give solutions within a multiplicative factor of 2 of the optimal value of d are known [12, 13], and a polynomial-time approximation scheme (PTAS) has also recently been developed [14]. Unfortunately, the high degree of the polynomial in the running time of the PTAS renders it of theoretical interest only. The dependence of the NP-hardness results [6, 12] on the hardness of **CLOSEST STRING**, a special case where $l = n$, indicates that this problem is worth investigating using fixed parameter techniques. The **CLOSEST STRING** problem has been shown to be fixed parameter tractable when d is fixed, by an algorithm that solves it in $O(nm + md \cdot d^d)$ time [9].

Table 1: Previously Known Exact Algorithms for 1AS.

Running Time / Space	Source
$O(nml\|\Sigma\|^l)$ time, $O(nm)$ space	Waterman *et al.* ('84) [24]
$O(nmlN(\Sigma,l,d)) = O(nml\|\Sigma\|^{d}l^{d+1})$ time, $O(nm\|\Sigma\|^l)$ space	Waterman *et al.* ('84) [24]
$O(nml\|\Sigma\|N(\Sigma,l,d)) = O(nm\|\Sigma\|^{d+1}l^{d+1})$ time, $O(nml)$ space	Sagot *et al.* ('95) [20]
$O(nm\log l\|\Sigma\|N(\Sigma,l,d)) = O(nm\log l\|\Sigma\|^{d+1}l^{d})$ time, $O(nmN(\Sigma,l,d)) = O(nm\|\Sigma\|^{d}l^{d})$ space	Sagot *et al.* ('95) [20]
$O(nm^2lN(\Sigma,l,d)) = O(nm^2\|\Sigma\|^{d}l^{d+1})$ time, $O(nml)$ space	Sagot ('98) [19]
$O(ml(n+\|\Sigma\|^l)) = O(nml + ml\|\Sigma\|^l)$ time, $O(mn)$ space	Blanchette *et al.* ('00) [2]
$O(nml + \|\Sigma\|^{2l})$ time	Evans and Wareham ('01) [5]
$O(nml + \|\Sigma\|^{l+d}l^{d})$ time	Evans and Wareham ('01) [5]
$O(mnl + nm\log n + ml\min(n-l,\|\Sigma\|^l)^m\|\Sigma\|^{dm+d}l^{d})$ time	Evans and Wareham ('01) [5]

2.1 Efficient Exact Solutions for 1AS

Exact FPT algorithms for 1AS are of particular interest in computational biology to find short segments that approximately occur in entire families of DNA or RNA sequences. These can be used as DNA sequence primers, as probes to detect sequence presence and distinguish sequences, as complementary sequences to block binding sites, and as other more general sequence family motifs. Typically these examples require only small parameter values. For example, instances of 1AS that occur in the design of DNA primers for groups of sequences in molecular biology have very small values for $|\Sigma|$, d, and l, *e.g.*, $|\Sigma| = 4$, $d \leq 3$, and $l \leq 25$ [5]. A general search for common sequence motifs has produced a challenge problem in the computational biology community – namely, 1AS when $n = 600$, $m \geq 15$, $|\Sigma| = 4$, $l = 15$, and $d = 4$ [17]. One result of this challenge is that 1AS is the only problem considered here that has a substantial body of previous algorithmic work. Much of this work is on heuristic algorithms and will not be discussed further here.

Explicit consideration of d-Hamming neighborhoods allows for better algorithms if one is willing to add d to the parameter. Indeed, almost all previous exact algorithm work on 1AS has proceeded along these lines. This work is summarized in Table 1. Each of the algorithms listed in Table 1 is preferable under a specific range of values for $|\Sigma|$, m, n, l, and d. However, the best algorithm overall is that by Sagot [19]. In this section, we introduce four algorithms that are fixed parameter tractable relative to various subsets of the aspects of 1AS. The first three are entirely new. The fourth algorithm, which uses an approach similar to that used in [19], runs in $O(nmlN(\Sigma,l,d)) = O(nm|\Sigma|^{d}l^{d+1})$ time and $O(nml)$ space.

Algorithm #1. 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; hence, the algorithm as a whole runs in $O(|\Sigma|^l mnl)$ time $O(mn)$ space. This is essentially the first algorithm given in [24]. The advantage of this approach is that the total space required is only a constant function of the input size.

Algorithm #2. Build the table over all strings of length l over Σ. For each string x in this table, build a set of all strings in $\mathcal{F}$ with a length-l substring in the d-Hamming neighborhood of x. These sets indicate the subset of $\mathcal{F}$ covered by x. At the end of this process, any length-l string over Σ that is marked for each string in $\mathcal{F}$ is a center for $\mathcal{F}$. As $N(\Sigma, l, d)$, the number of strings in the d-Hamming neighborhood of a length-l string over Σ, is $O(|\Sigma|^d l^d)$ [21, Theorem 6], the table and substrings can be built in $O(m|\Sigma|^l N(\Sigma, l, d)) = O(m|\Sigma|^l|\Sigma|^d l^d) = O(|\Sigma|^{d+l} l^d m)$ time. Hence, the algorithm as a whole runs in $O(nml + m|\Sigma|^{d+l} l^d)$ time and $O(m|\Sigma|^l)$ space. In comparison with Algorithm #1, this algorithm eliminates multiplicative dependence between n and Σ^l. In fact, aside from reading the input, there is no dependence on n in the time or space complexities, i.e., this is an *on-line* algorithm.

Algorithm #3. We introduce a new character $x \notin \Sigma$, called the *blocking character*. The importance of x is that it will always induce a mismatch when compared to a character in $\mathcal{F}$. Let s be a length-l string containing at most d occurrences of the character x. Let $S_i[j]$ be a length-l substring of $S_i \in \mathcal{F}$. A substitution of s under $S_i[j]$, denoted $sub(s, S_i[j])$, is a replacement of a subset of the occurrences of x in s with the characters at corresponding positions in $S_i[j]$. A *minimal matching* substitution, denoted $minsub(s, S_i[j])$, is a substitution that results in $dist_H(s, S_i[j]) \leq d$ having the additional property that no substitution replacing *a subset of those same positions* results in $dist_H(s, S_i[j]) \leq d$ (note that a minimal matching substitution need not be unique).

The algorithm is based on the observation that to find a center, it is sufficient to obtain an instance of the center and change characters in at most d positions of the instance. The first step is to isolate a string, S', in which we attempt to find an instance of a center. For each length-l substring of S', and each size d set of positions in S', change the characters at those positions to x. Then execute the procedure *DevelopCenter*($\mathcal{F} \setminus S'$, $S'[j]$).

Consider the search space of this algorithm as corresponding to the recursion tree of the procedure *DevelopCenter* (see Table 2). The time complexity of this 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 $n\binom{d}{d/2}$, corresponding to the maximum number of substrings that must must be tried, multiplied by the maximum number of minimal matching substitutions that must be tried. The maximum number of leaves in the search space is $n\binom{l}{d}(n\binom{d}{d/2})^d$ and $O(nm)$ time is required for each leaf. The complexity of this algorithm is bounded by $O(nm(\binom{l}{d}\binom{d}{d/2}n)^d)$. Of note is the absence of any parameter representing the alphabet (nm or Σ).

Algorithm #4. This is an improvement of an efficient algorithm due to Sagot [19]. The algorithm begins by constructing, for each $S_i \in \mathcal{F}$, the lexicographic tree T_i of length-l substrings of S_i. This requires $O(nml)$ time. The centers are searched enumeratively by traversing the space of all possible centers. The centers of desired length l are not searched directly. The search process iteratively searches for each prefix of a given center in order to

Table 2: Procedure *DevelopCenter*

```
      procedure DevelopCenter(F, C)

1     Let S be an arbitrary string in F; branch ← true;
2.    for all 1 ≤ j ≤ n−l+1 do
3.       if dist_H(S[j], C) ≤ d then
4.          branch ← false
5.          DevelopCenter(F \ S, C)
6.    if branch = true then
7.       for all 1 ≤ j ≤ n−l+1 do
8.          if dist_H(S[j], C) ≤ 2d then
9.             for each C' ← minsub(C, S[j]) do
10.               DevelopCenter(F \ S, C')
```

take advantage of the fact that prefixes are shared by many potential centers and eliminate redundant processing.

Let α be a (length $\leq l$) center for $\mathcal{F}$. Define $fron^\alpha = \{fron_1^\alpha, \ldots, fron_m^\alpha\}$. For each $1 \leq i \leq m$, $fron_i^\alpha = \{(v, v_{err}) : v \in nodes(T_i), v_{err} \leq d\}$. Think of $fron_i^\alpha$ as the frontier of nodes in T_i whose path labels are of Hamming distance $\leq d$ from α. For any $(v, v_{err}) \in fron_i^\alpha$, the path label of node v spells out an instance of α.

While searching the space of possible centers, if any $fron_i^\alpha \in fron^\alpha$ is found to be empty, the search space is pruned. This condition implies that there exists some member of $\mathcal{F}$ containing no instance of α, the current center. Pseudocode for this algorithm is provided in Table 3. For the initial call to *FindCenters*, α is the empty string and $fron^\alpha$ is the set of roots of T_i with an error term of 0.

The time complexity of the algorithm is proportional to the number of centers in the search space multiplied by the size of the $fron$ set that must be constructed for each iteration of the search. In the worst case, for m strings of length n, there will be $O(nN(\Sigma, l, d))$ potential centers of length l with $dist_H \leq d$ from any substring of a member of $\mathcal{F}$. Let the (d, l)**-neighborhood of a string** S refer to the set of strings within the d-Hamming neighborhood of any length-l substring of S. The maximum size of the (d, l)-neighborhood of a string S of length n is $(n - l + 1)N(\Sigma, l, d)$, and this is achieved when the d-Hamming neighborhoods of all length-l substrings of S are disjoint. Further observe that the upper bound of $(n - l + 1)N(\Sigma, l, d)$ on the center search space can only occur when all (d, l)-neighborhoods of members of $\mathcal{F}$ completely overlap. Attaining this limit on the search space requires that each $fron_i^\alpha \in fron^\alpha$ have only one element, *i.e.*,

$$\begin{aligned} \text{Worst case running time} &= (\text{Size of Search Space}) \times (\text{Size of } fron) \\ &= O(nlN(\Sigma, l, d)) \times O(m). \end{aligned}$$

Table 3: Procedure *FindCenters*.

```
      procedure FindCenters(α, fron^α)

1.    for each character σ ∈ Σ do
2.       for each set fron_i^α ∈ fron^α do
3.          for each pair (v, v_err) ∈ fron_i^α do
4.             if there is an arc (v, v') in T_i labelled with σ then
5.                add (v', v_err) to fron_i^ασ
6.             if (v_err < d) then
7.                for all arcs in T_i from v to some v' labelled with σ' ≠ σ do
8.                   add (v', v_err + 1) to fron_i^ασ
9.       if no member fron_i^ασ ∈ fron^ασ is empty then
10.         if (|ασ| = l) then
11.            print out center ασ
12.         else
13.            FindCenters(ασ, fron^ασ )
```

Table 4: A Summary of the 1AS Algorithms Derived in this Paper.

Alg #	Running Time	Space
1	$O(nml\|\Sigma\|^l)$	$O(nm)$
2	$O(nml + m\|\Sigma\|^{d+l}l^d)$	$O(m\|\Sigma\|^l)$
3	$O(m2^{d^2}l^d n^{d+1})$	$O(nm)$
4	$O(nm\|\Sigma\|^d l^{d+1})$	$O(nml)$

Consider the effect on the search space of any $fron_i$ having more than one element. The d-Hamming neighborhoods of substrings of S_i would no longer be disjoint and the (d,l)-neighborhood of S_i would have at least one fewer member, thus eliminating a node from the search space. Hence, the overall running time of the algorithm is $O(nmlN(\Sigma,l,d)) = O(nm|\Sigma|^d l^{d+1})$.

The algorithm in [19] actually solves a generalized "quorum" version of 1AS that searches for strings in Σ^l that are within Hamming distance d of substrings in at least $q \leq m$ (as opposed to m) of the given strings. We have improved on [19] by replacing the traversal of a single generalized suffix tree with the simultaneous traversal of a set of lexicographic trees.

Table 5: Known fixed parameter tractable aspects of 1AS

Parameter	–	Σ	m	m, Σ
–				
d				
l		FPT		FPT
l, d		FPT		FPT
n	FPT	FPT	FPT	FPT
n, d	FPT	FPT	FPT	FPT
n, l	FPT	FPT	FPT	FPT
n, l, d	FPT	FPT	FPT	FPT

2.2 Parameterized Complexity of 1AS

We begin by identifying the combinations of parameters that are sufficient to render 1AS fixed parameter tractable. We present two results that anchor all known fixed parameter tractability results.

Theorem 2.1 *Fixed Parameter Tractability of* APPROXIMATE SUBSTRING.

1. $1\text{AS}(\Sigma, l) \in FPT$.
2. $1\text{AS}(n) \in FPT$.

Proof (1). Follows from Algorithm #1 above, which solves 1AS in time $O(nml|\Sigma|^l)$.
Proof (2). Consider Algorithm #3 with time complexity $O(m2^{d^2}l^d n^d{+}1)$. Since $m2^{d^2}l^d n^{d+1} = O(mn^{O(n^2)})$, $1\text{AS}(n)$ is in FPT. ∎

The picture we have so far of the complexity of **1AS** is shown in Table 5. The results in the bottom half of the table are obtained by noting that the result of Theorem 2.1 where n is fixed applies to all of these. Similarly for the entries where $|\Sigma|$ and l are fixed. The algorithms used in the proof of Theorem 2.1 anchor the known fixed parameter tractability results for 1AS.

The following problems will serve as source problems to demonstrate hardness:

CLIQUE [7, Problem GT19]

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

Parameter: A positive integer k.

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

DOMINATING CLIQUE [4, Page 463]

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

Parameter: A positive integer k.

Question: Is there a set of k vertices $V' \subseteq V$ that form a complete subgraph of G(that is, a clique of size k) and also form a dominating set for G?

To show W[1]-hardness for $1\text{AS}(m, l, d)$ we reduce from CLIQUE. Let $G = (V, E)$ be a graph for which we wish to determine whether G has a k-clique. We show how to construct a family $\mathcal{F}_G$ 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 $V = \{1, \ldots, |V|\}$.

Target Parameters. The number of strings in $\mathcal{F}_G$ is $m = f_1(k) = \binom{k}{2}$. The length of center $\mathcal{C}_G$ is $l = f_2(k) = k + 2$, and the maximum distance between instance and center is $d = f_3(k) = k - 2$. The maximum length of any string in $\mathcal{F}_G$ (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).

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

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:

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

Separators:

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

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

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

String S_{ij} is composed of all edge components from $\mathcal{E}_{ij}$ arranged in the following manner (where product notation refers to concatenation):

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

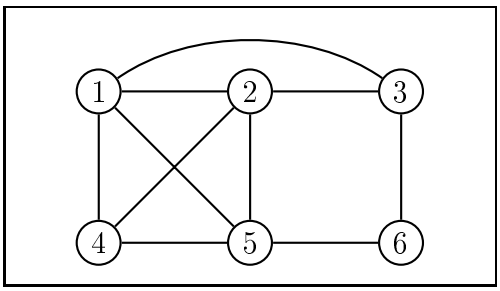

Figure 1: Graph 1

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}:	$Auu36Bu^6Auu45Bu^6Auu56B$

Figure 2: $\mathbf{1AS}(m, l, d)$ representation for Graph 1 (desired clique size $k = 4$).

An example of the reduction for the graph in Figure 1 and a desired clique size 4 can be seen in Figure 2. It is evident from the example that any center for $\mathcal{F}_G$ will have the property that all positions other than the terminal positions will be occupied by vertex characters in ascending order (this will be proven below).

Preliminary Results. Some additional conventions are used in discussing $\mathcal{F}_G$. An instance 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 instances 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 instances can come from the same string.

Lemma 2.1 *Let $\mathcal{C}_G$ be a center for $\mathcal{F}_G$. The following are true:*

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

Proof (1). Suppose $\mathcal{C}_G$ begins with a character other than A. Then the separation between A and B in members of $\mathcal{F}_G$ prevents any instance from matching both A and B in $\mathcal{C}_G$. In order to match $\mathcal{C}_G$ at 4 positions, each instance must then match in a position occupied by a character from Σ_2. By the pigeonhole principle, this results in a contradiction.

Proof (2). Suppose $\mathcal{C}_G$ contains a character from Σ_2 in position z. Then at most one instance matches $\mathcal{C}_G$ at position z. Consider the vertex group $\mathcal{V}_z$. Any instance from $\mathcal{V}_z$ that matches $\mathcal{C}$ out-of-phase must determine a unique character, since it can't match both A and B. Suppose some instance 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 instances from $\mathcal{V}_z$ determine unique characters, and $|\mathcal{V}_z| = k-1$ at most one instance can match $\mathcal{C}$ at 4 positions, the pigeonhole principle once again presents a contradiction.

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

Proof (4). Suppose $\mathcal{C}_G$ has been partially determined by instances from $\mathcal{V}'_z \subset \mathcal{V}_z$, for vertex position z. Consider instance $\mathcal{I}$ from $S_{z,x} \in \mathcal{V}_z \setminus \mathcal{V}'_z$. 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 $\mathcal{C}_G$, $\mathcal{I}$ must determine position x in $\mathcal{C}_G$. The first instance determines 4 positions in the center, and the remaining $k-2$ instances each determine an additional position, a total of $k+2$ positions. ∎

Lemma 2.2 CLIQUE $\leq_m$ **1AS**(m, l, d).

Proof. 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 $\mathcal{F}_G$. Conversely, suppose there is no k-clique in G and there is a center $\mathcal{C}_G$ for those strings in vertex group $\mathcal{V}_z$ where vertex v occupies vertex position z. By Part (4) of the lemma, instances from $\mathcal{V}_z$ completely determine $\mathcal{C}_G$. 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 $\mathcal{F}_G$, 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 $\mathcal{C}_G$ is not a center for $\mathcal{F}_G$. ∎

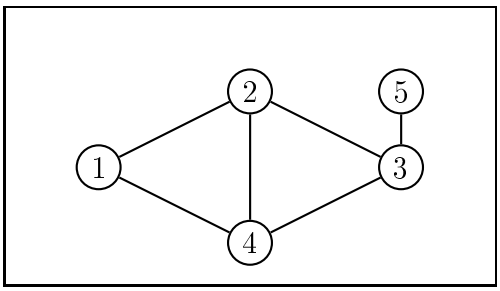

Figure 3: Graph 2.

Next we investigate the complexity of 1AS when parameterized by l and d. To show W[2]-hardness, we reduce from the W[2]-complete problem DOMINATING CLIQUE [4]. Let $G = (V, E)$ be a graph for which we wish to determine whether G has a dominating clique of size k. We show how to construct a family $\mathcal{F}_G$ 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 $V = \{1, \ldots, x\}$. The alphabet and substring gadgets are exactly the same as for the previous reduction.

The Target Parameters. The number of strings in $\mathcal{F}_G$ 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 above.

The Reduction. The strings will form two groups $\mathcal{F}_G = \mathcal{F}_{G_E} \cup \mathcal{F}_{G_V}$ 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 instances from $\mathcal{F}_{G_E}$ corresponds not only to a k-clique, but to a dominating set as well:

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

String S_{Vp} 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):

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

An example of this reduction for the graph in Figure 3 and a desired dominating clique size of 3 can be seen in Figure 4.

Lemma 2.3 DOMINATING CLIQUE $\leq_m$ 1AS(l, d).

S_{12}: $A12uBu^5A14uBu^5A23uBu^5A24uBu^5A34uB$
S_{13}: $A1u4Bu^5A2u4Bu^5A3u5B$
S_{23}: $Au23Bu^5Au24Bu^5Au34Bu^5Au35B$

S_{V_1}: $A12uBu^5A14uBu^5A23uBu^5A24uBu^5A34uB$
$A1u4Bu^5A2u4Bu^5Au23Bu^5Au24Bu^5Au34B$
S_{V_2}: $A12uBu^5A14uBu^5A23uBu^5A24uBu^5A34uB$
$A1u4Bu^5A2u4Bu^5A3u5Bu^5Au23Bu^5Au24B$
$Au34Bu^5Au35B$
S_{V_3}: $A12uBu^5A14uBu^5A23uBu^5A24uBu^5A34uB$
$A1u4Bu^5A2u4Bu^5A3u5Bu^5Au23Bu^5Au24B$
$Au34Bu^5Au35B$
S_{V_4}: $A12uBu^5A14uBu^5A23uBu^5A24uBu^5A34uB$
$A1u4Bu^5A2u4Bu^5A3u5Bu^5Au23Bu^5Au24B$
$Au34Bu^5Au35B$
S_{V_5}: $A23uBu^5A34uBu^5A3u5Bu^5Au23Bu^5Au34B$
$Au35B$

Figure 4: $\mathbf{1AS}(l,d)$ representation for Graph 2 (with desired dominating clique size $k=3$).

Proof. As was shown in Theorem 1, 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_{Vp} that encodes any of the $k-1$ clique edges incident on vertex p will serve as an instance for the center. Therefore a dominating k-clique in G implies a center for $\mathcal{F}_G$. 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 $S_{Vp} \in \mathcal{F}_{G_V}$, none will correspond to an edge in the clique and therefore none will match the center sufficiently to be an instance. ∎

The following result of Francis and Litman [6] is important in establishing the hardness a version of 1AS. The result will be used again in Section 4.2 to establish the hardness of many versions of pAS.

Theorem 2.2 *[6]* pAS *is NP-hard when* $|\Sigma| = 2$, $p = 1$, *and* $l = n$.

Theorem 2.3 *Hardness of* APPROXIMATE SUBSTRING*:*

1. $1\mathbf{AS}(m,l,d)$ *is hard for* $W[1]$.
2. $1\mathbf{AS}(l,d)$ *is hard for* $W[2]$.
3. $1\mathbf{AS}(\Sigma)$ *is not in XP unless* $P = NP$.

Proof (1). This follows from Lemma 2.2 and the $W[1]$-hardness of CLIQUE [4].
Proof (2). This follows from Lemma 2.3 and the $W[2]$-hardness of DOMINATING CLIQUE [4].
Proof (3). This follows immediately from Theorem 2.2, which demonstrated that 1AS is NP-hard when $|\Sigma| = 2$. ∎

Now we show inclusion in classes of the W-hierarchy for restricted aspects of 1AS. The idea is to use a truth assignment of weight d to indicate the characters occupying the l center positions. This circuit will be called the *center testing circuit.*

Center Testing Circuit: Let $\mathcal{F} = \{S_1, \ldots, S_m\}$ be an instance of 1AS, and $\mathcal{C}$ is any center for $\mathcal{F}$. The j-th length-l substring of S_i will be denoted $S_i[j]$. The set X will be used to index size $l-d$ subsequences of a length-l string:

$$X = \{X_p : X_p \subset \{1, \ldots, l\}, |X_p| = l - d, 1 \leq p \leq \tbinom{l}{d}\}.$$

Let $A = \{a[i, j, p, q] : \ 1 \leq i \leq m, \ 1 \leq j \leq n-l+1, \ 1 \leq p \leq |X|, \ 1 \leq q \leq l-d\}$ 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 $a[i, j, p, q]$ will take on 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 $S_1 = tggtca$, $S_2 = accgac$, and $S_3 = cggtag$ over alphabet $\Sigma = \{a, c, g, t\}$. We assume the order $a = 1$, $c = 2$, $g = 3$ and $t = 4$ on Σ. If $p = (1, 2, 3)$, 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 and if $p = (1, 3, 5)$, then $a[2, 2, p, 2] = b[4, 3]$ 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.

• *Proof of Correctness.* 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, it is easy to verify that a truth assignment corresponding to $\mathcal{C}$ 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 $s \in \Sigma^l$. The clauses of E_2 ensure that

for each i, some substring $S_i[j]$ matches $l-d$ positions of s. This implies that in each S_i, there is a substring of length-l that is distance less than d from s. Therefore s is a center for $\mathcal{F}$.

Instance Testing Circuit. We construct a new circuit, called the *instance testing circuit*, having little resemblance to the center testing circuit. Our goal here is to show membership for versions of 1AS when l is left free. The idea this time is to select m instances and, for each instance, d positions where the instance 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 $b[i,j]$ will be set true when $S_i[j]$ is an instance 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 instance 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 d for any instance. In the description of the circuit, the set of variables $A = \{a[i,j,p,q] : 1 \leq i \leq m,\ 1 \leq j \leq n-l+1,\ 1 \leq p \leq l,\ 1 \leq q \leq |\Sigma|\}$ will act as an alias for the variables from B. For any occurrence 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 $E = E_1E_2E_3$ be the boolean expression over the set of variables of $B \cup W$, where:

$$E_1 = \prod_{i=1}^{m} \prod_{1 \leq j < j' \leq n-l+1} (\neg b[i,j] + \neg b[i,j']),$$

$$E_2 = \prod_{i=1}^{m} \prod_{r=1}^{d} \prod_{1 \leq p < p' \leq l} (\neg w[i,r,p] + \neg w[i,r,p']),$$

$$E_3 = \prod_{p=1}^{l} \sum_{q=1}^{|\Sigma|} \prod_{i=1}^{m} \Big(\sum_{r=1}^{d} w[i,r,p] + \sum_{j=1}^{n-l+1} a[i,j,p,q] \Big).$$

For example consider the set of strings $S_1 = tggtca$, $S_2 = accgac$, and $S_3 = cggtag$. If $p = (1,2,3)$, 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 and if $p = (1,3,5)$, then $a[2,2,p,2] = b[4,3]$ corresponding to g at position 4.

• *Proof of Correctness.* 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 instance $S_i[j]$ of $\mathcal{C}$, and also setting $w[i,r,p]$ to true if the r-th mismatch in the instance from S_i occurs at position p. This can be easily verified. 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 instances, one from each S_i. The clauses of E_2 ensure that at most d mismatching positions are selected for the instance from any S_i. E_1 and E_2 combined force $\mathcal{T}$ to correspond directly to a set of instances and a

set of positions where each instance may differ from a center. The fact that $\mathcal{T}$ satisfies E_3 implies that all instances agree in all positions with the possible and permitted exception of the exempted positions. Hence $\mathcal{F}$ has a center. ∎

Single Instance + 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 $S_1[j]$, then guess the positions and characters by which the center differs from $S_1[j]$. 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 inputs:

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

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 instance and are not part of a truth assignment). The weight $d+1$ truth assignment will come from selecting exactly one member of X_2 (representing a substring of S_1) and d members of X_3(representing the substitutions). Once the center has been "guessed", it remains to test it against potential instances from the other strings in $\mathcal{F}$. Unlike the **center testing circuit** above, l is not fixed, so we cannot use the same strategy 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

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

The lower layers of the circuit are described by the variables:

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

with the interpretation that $b[i,j,p] = true$ 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 $b[i,j,p]$ depends on variables used to generate $b[i,j,p-1]$. The purpose of this is to prevent having to count the number of mismatches (between the guessed center and an instance) 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 the counter for each $S_i[j]$:

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

such that

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

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:

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

The circuit C is described by expression $E = E_1 E_2 E_3$ defined as:

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

• *Proof of Correctness.* It is easily verified that 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 $O(nml(|\Sigma|+d))$. The depth of the circuit is $O(l)$ since, for each i,j and r, there is a path passing through $a[i,j,1,r]\,, a[i,j,2,r] \ \ldots \ a[i,j,l,r]$. ∎

Theorem 2.4 *Membership of* **APPROXIMATE SUBSTRING** *in classes of the W-hierarchy:*

1. $1\textsc{as}(l) \in W[2]$.
2. $1\textsc{as}(m,l) \in W[1]$.
3. $1\textsc{as}(m,d) \in W[3]$.
4. $1\textsc{as}(m,\Sigma,d) \in W[2]$.
5. $1\textsc{as}(d) \in W[P]$.

Proof (1). This follows trivially by observing that when l is fixed, the center testing circuit has weft 2.

Proof (2). If m is fixed along with l, the center testing circuit has weft 1.

Table 6: The Parameterized Complexity of the 1AS Problem.

Parameter	–	Σ	m	m, Σ
–	NP-Complete	$\notin XP$	$W[1]$-Hard	
d	$W[2]$-Hard, in $W[P]$	in $W[P]$	$W[1]$-Hard, in $W[3]$	in $W[2]$
l	$W[2]$-Complete	FPT	$W[1]$-Complete	FPT
l, d	$W[2]$-Complete	FPT	$W[1]$-Complete	FPT
n	FPT	FPT	FPT	FPT
n, d	FPT	FPT	FPT	FPT
n, l	FPT	FPT	FPT	FPT
n, l, d	FPT	FPT	FPT	FPT

Proof (3). This follows trivially by observing that when m and d are fixed, the instance testing circuit has weft 3.

Proof (4). This follows from (3) because fixing $|\Sigma|$ reduces the weft of the instance testing circuit by one.

Proof (5). When d is fixed, and all other parameters left free, the single instance + modification circuit has weft $O(l)$. ∎

3 p-EXACT SUBSTRING

For many families of sequences, there may not be a single substring that is contained in each sequence. When no single substring is found in all sequences, it is useful to investigate whether there can be a set of substrings that between them are found in all sequences. This problem is applicable in computational biology to characterize more diverse sequence families, and to find a small set of probes that would detect an entire group of DNA sequences. No results prior to those in this paper are known for this problem.

3.1 Efficient Exact Solutions for pES

Algorithm #5. Consider the algorithm that first determines, for each length-l substring of a string in $\mathcal{F}$, the subsets of $\mathcal{F}$ whose strings have that substring, and then checks each of the possible p-partitions of $\mathcal{F}$ to see if each subset in that partition is covered by some length-l substring. Finding the sets of $\mathcal{F}$ covered by length-l substrings of $\mathcal{F}$ can be done in $O(mn)$ time using a generalized suffix tree [10, Section 6.4]. Marking the presence or absence of each possible subset of $\mathcal{F}$ can be done in $O(m2^m)$ time using a table with 2^m 1-bit entries. Since each of the $O(2^m)$ tree-derived subsets has at most $m-1$ immediate supersets containing exactly one more element, it will be added at most once for itself and once per immediate superset. As there are p^m partitions of $\mathcal{F}$, each of which can be checked in $O(mp)$ time, the the algorithm as a whole runs in $O(nm + m2^m + mp^{m+1})$ time and $O(2^m)$ space.

Algorithm #6. Consider the following algorithm based on the classical pseudo-polynomial time algorithm for INTEGER KNAPSACK [15, Section 16.2]: Let $f : 2^{\mathcal{F}} \rightarrow \{0, \ldots, 2^m\}$ be the one-to-one correspondence that associates each subset of $\mathcal{F}$ with the integer value corresponding to the m-bit representation of that subset. Determine the set C of subsets of $\mathcal{F}$ covered by the length-l substrings of the strings in $\mathcal{F}$, construct the directed graph $G = (V, A)$ where $V = \{0, \ldots, 2^m\}$, $A = \{(i, j) : j = i \ \vee \ f(c) \text{ for some } c \in C\}$, and $\vee$ is the bit-wise OR operation, and use breadth-first search from vertex 0 to determine the length of the shortest path from 0 to 2^m. In this graph, vertices correspond to subsets of $\mathcal{F}$ and each edge (i, j) encodes the fact that subset j of $\mathcal{F}$ can be obtained by the union of subset i of $\mathcal{F}$ and some set in C. Note that a path from vertex 0 to vertex i corresponds to a subset of C whose union is subset i of $\mathcal{F}$; hence, there is a p-Center for $\mathcal{F}$ if and only if there is a path of length p from vertex 0 to vertex 2^m in G. Set C can be computed in $O(nm + 2^m)$ time and $O(2^m)$ space as in part (3) above, graph G can be constructed in $O(|V| + |A|) = O(2^m + 2^m|C|) = O(\min(nm, 2^m)2^m)$ time and space, and a breadth-first search can be done in $O(|V| + |A|) = O(\min(nm, 2^m)2^m)$ time and space. Hence, the algorithm as a whole runs in $O(nm + \min(nm, 2^m)2^m)$ time and space.

Algorithm #7. Examine each p-selection of the $|\Sigma|^l$ strings of length l over Σ to see if they form a set of centers for $\mathcal{F}$. As there are $\binom{|\Sigma|^l}{p} = O(|\Sigma|^{lp})$ such p-selections, each of which can be checked in $O(nmp)$ time, the algorithm as a whole runs in $O(nmp|\Sigma|^{lp})$ time.

Algorithm #8. Determine, for each of the $|\Sigma|^l$ strings of length l over Σ, the subset of $\mathcal{F}$ whose strings have that substring, and then check each p-selection of these strings to see if they form a set of centers for $\mathcal{F}$. The first step can be done in $O(nml + m|\Sigma|^l)$ time (initialize the table to empty and then mark each length-l substring appropriately in a one-pass scan over the strings in $\mathcal{F}$). As there are $O(|\Sigma|^{lp})$ p-selections, each of which can be checked in $O(mp)$ time, the algorithm as a whole runs in $O(nml + mp|\Sigma|^{lp})$ time and $O(m|\Sigma|^l)$ space.

Algorithm #9. Consider the following algorithm based on the search-tree algorithm for VERTEX COVER [4]. Each node in this search tree is labeled with subsets of $\mathcal{F}$. The root node is labeled with the set $\mathcal{F}$, and the tree is constructed recursively for each node v with associated set $\mathcal{F}'$ as follows:

1. If $\mathcal{F}' = \phi$, v is a leaf in the search tree.
2. If $\mathcal{F}' \neq \phi$, select an arbitrary string S from $\mathcal{F}'$ and create $(n - l) + 1$ children of v such that the i-th child node, $1 \leq i \leq (n - l) + 1$, is labeled with the subset of $\mathcal{F}'$ that does not contain the length-l substring of S beginning at position i.

Note that any leaf in the search tree of depth p corresponds to a set of $\leq p$ substring centers for $\mathcal{F}$, and if there is no such leaf, there is no solution for the given instance of pES. As, the search tree contains at most $(n - l + 1)^p - 1$ nodes, each of which requires $O(m)$ space and can be created in $O(nm)$ time, the algorithm as a whole runs in $O(mln^{p+1})$ time and $O(mn^p)$ space.

Table 7: A Summary of p**ES** Algorithms Derived in this Paper.

Alg #	Running Time	Space
5	$O(nm + m2^m + mp^{m+1})$	$O(2^m)$
6	$O(nm + \min(nm, 2^m)2^m)$	$O(nm + \min(nm, 2^m)2^m)$
7	$O(nmp\|\Sigma\|^{lp})$	$O(nm)$
8	$O(nml + mp\|\Sigma\|^{lp})$	$O(m\|\Sigma\|^l)$
9	$O(mln^{p+1})$	$O(mn^p)$

Table 8: Fixed parameter tractable aspects of p**ES**

Parameter	–	Σ	m	m, Σ
–			FPT	FPT
l		FPT	FPT	FPT
n		FPT	FPT	FPT
n, l		FPT	FPT	FPT
p			FPT	FPT
p, l		FPT	FPT	FPT
p, n	FPT	FPT	FPT	FPT
p, n, l	FPT	FPT	FPT	FPT

3.2 Parameterized Complexity of pES

Theorem 3.1 *Fixed Parameter Tractability of* p-**EXACT SUBSTRING**:

1. $p\mathbf{ES}(m) \in FPT$
2. $p\mathbf{ES}(p, n) \in FPT$
3. $p\mathbf{ES}(\Sigma, l) \in FPT$.

Proof (1). Follows from Algorithm #6 in section 3.1 which has time complexity $O(mn+4^m)$.
Proof (2). Follows from Algorithm #9 in section 3.1, which has time complexity $O(mn^{O(p)})$.
Proof (3). Follows from Algorithm #7 in section 3.1, and the observation that $p \leq |\Sigma|^l$. ∎

The results of Theorem 3.1 (and all results that follow by inheritance) are presented in Table 8. In the rest of this section we show that unless $P = NP$, Table 8 actually represents all fixed parameter tractability results for p**ES**. No hardness results prior to those in this paper are known for this problem. All hardness results in this section will be derived via reductions from the following problems:

VERTEX COVER [7, Problem GT2]

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

Parameter: 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'?

HITTING SET [7, Problem SP8]

Instance: A collection C of subsets of a finite set S and a positive integer $k \leq |C|$.

Parameter: A positive integer k.

Question: Is there a subset of $S' \subseteq S$ with $|S'| \leq k$ such that S' contains at least one element from each subset in C?

Lemma 3.1 **VERTEX COVER** $\leq_m$ p**ES**, *for any* $n \geq 2$ *and* $l \geq 1$.

Proof. Given an instance $\langle G, k \rangle$ of **VERTEX COVER**, construct the following instance $\langle \Sigma', S', p', l' \rangle$ of p**ES**: Let $\Sigma' = V$, $S = \{uv \mid (u, v) \in E\}$, $p' = k$, and $l' = 1$. Note that in the constructed instance of p**ES**, $p' = k$, $n = 2$ and $l = 1$. ∎

Lemma 3.2 **HITTING SET** $\leq_m$ p**ES**(p, l).

Proof. Given an instance $\langle S, C, k \rangle$ of **HITTING SET**, construct the following instance $\langle \Sigma', S', p', l' \rangle$ of p**ES**(p, l): Let $\Sigma' = S$, $S' = \{c_1 c_2 \ldots c_{|c|} \mid c = \{c_1, c_2, \ldots, c_{|c|}\} \in C\}$, $p' = k$, and $l' = 1$. Note that in the constructed instance of p**ES**(p, l), $p = k$ and $l = 1$. ∎

Lemma 3.3 **HITTING SET** $\leq_m$ p**ES**(Σ, p), *for any* Σ *with* $|\Sigma| \geq 2$.

Proof. Given an instance $\langle S, C, k \rangle$ of **HITTING SET** and any alphabet Σ such that $|\Sigma| \geq 2$, construct the following instance $\langle \Sigma, S', p', l' \rangle$ of p**ES**(Σ, p): Let $f : S \to \{1, 2, \ldots, |S|\}$ be a one-to-one correspondence that induces an order on S and let a and b be any pair of distinct elements of Σ. For any $x \in S$, let w_x be the string obtained from the length $d = \lceil log_2(|S|) \rceil$ bit-wise representation of $f(x)$ by substituting a for 0 and b for 1. Let

$$S' = \{b^{d+1}a^{d+1}w_{c_1}a^{d+1}b^{d+1}a^{d+1}w_{c_2}a^{d+1}b^{d+1}\ldots a^{d+1}w_{c_{|c|}}a^{d+1}b^{d+1}a \mid c = \{c_1, c_2, \ldots, c_{|c|}\} \in C\},$$

$p' = k$, and $l = 5d + 5$. If a set $c \in C$ contains an element x, its corresponding string s_c in S' will contain the substring $b^{d+1}a^{d+1}w_x a^{d+1}b^{d+1}a$ of length $5d + 5$. Moreover, any center

string of length $5d + 5$ for a subset of S' will have a substring of the form aw_ja for some $1 \leq j \leq |S|$, and this substring will occur in a string $s_c \in S'$ if and only if $j \in c$. Note that $p' = k$ in the constructed instance of $p\textbf{ES}(\Sigma, p)$. ∎

Note that none of these reductions changes the cost of a solution (since $p' = k$ in all cases), so they are also L-reductions [16] that preserve polynomial-time approximability.

Theorem 3.2 *Hardness of* p-**EXACT SUBSTRING**:

1. $p\textbf{ES}(p, l)$ *is* $W[2]$*-hard.*
2. $p\textbf{ES}(\Sigma, p)$ *is* $W[2]$*-hard.*
3. $p\textbf{ES}(n, l)$ *is not in* XP *unless* $P = NP$.
4. $p\textbf{ES}(\Sigma)$ *is not in* XP *unless* $P = NP$.

Proof (1). This follows from Lemma 3.2 and the $W[2]$-hardness of **HITTING SET** [4].
Proof (2). This follows from Lemma 3.3 and the $W[2]$-hardness of **HITTING SET** [4].
Proof (3). This follows from Lemma 3.1 and the NP-hardness of **VERTEX COVER** [7].
Proof (4). This follows from Lemma 3.3 and the NP-hardness of **HITTING SET** [7]. ∎

Theorem 3.3 $p\textbf{ES}(p) \in W[2]$.

Proof. We use a simple reduction to **HITTING SET**. Given instance $\langle \mathcal{F}, l, p \rangle$ of $p\textbf{ES}(p)$, construct the following instance $\langle S, C, k \rangle$ of **HITTING SET**: Let S be a set with elements corresponding to length-l substrings of members of $\mathcal{F}$, let $C = \{C_1, \ldots, C_m\}$ such that C_i is the subset of S with members corresponding to substrings of $S_i \in \mathcal{F}$, and let $k = p$. The correctness of the reduction is easily verified. The membership of **HITTING SET** in $W[2]$ [4] implies the membership of $p\textbf{ES}(p)$. ∎

Corollary 3.4 *There is no polynomial time approximation algorithm relative to* p *for* $p\textbf{ES}$ *within a logarithmic factor (*$c \log p$ *for any constant* c*) when* $|\Sigma| \geq 2$ *unless* $P = NP$.

Proof. Raz and Safra [18] showed that **SET COVER** cannot be approximated relative to k in polynomial time within a logarithmic factor unless $P = NP$. As the **HITTING SET** and **SET COVER** problems are equivalent [1] and **HITTING SET** is L-reducible to $p\textbf{ES}$ when $|\Sigma| \geq 2$ by Lemma 3.3, $p\textbf{ES}$ has the same restrictions on approximability. ∎

Table 9: The Parameterized Complexity of p**ES**

Parameter	–	Σ	m	m, Σ
–	NP-Complete	$\notin XP$	FPT	FPT
l	$\notin XP$	FPT	FPT	FPT
n	$\notin XP$	FPT	FPT	FPT
n, l	$\notin XP$	FPT	FPT	FPT
p	$W[2]$-Complete	$W[2]$-Complete	FPT	FPT
p, l	$W[2]$-Complete	FPT	FPT	FPT
p, n	FPT	FPT	FPT	FPT
p, n, l	FPT	FPT	FPT	FPT

4 p-APPROXIMATE SUBSTRING

Just as p**ES** can be applied to find a set of strings that at least one from the set exactly occurs in each sequence, p**AS** can be used to find a set of strings such that at least one from the set approximately occurs in each sequence. The set of strings will cover all sequences between them. For small values of d, p**AS** can also be applied to finding a set of DNA probes, or a motif for a large and more general family of sequences. There has been no work done directly on this problem prior to that reported in this paper. However, various results are implicit in results derived for restricted versions of this problem. The NP-hardness results for 1**AS** given in [6, 12] trivially imply the NP-hardness of p**AS**. Similarly, the inapproximability of p**AS** relative to d when $l = n$ and $|\Sigma| = 2$ within a multiplicative factor of $c < 2$ unless $P = NP$ [8, Theorem 6] implies the same for p**AS**.

4.1 Efficient Exact Solutions for pAS

Algorithm #10. Modify Algorithm #7 from Section 3.1 by replacing the $(n + l)$ term for simple pattern matching with an nl term for computation of Hamming distance, as in Algorithm #1 from Section 2.1. The resulting algorithm runs in $O(nmlp|\Sigma|^{lp})$ time.

Algorithm #11. Modify Algorithm #8 from Section 3.1 to use the table computed in Algorithm #2 from Section 2.1. The resulting algorithm runs in $O(nml + m|\Sigma|^{2l}l^l + mp|\Sigma|^{lp})$ time and $O(m|\Sigma|^l)$ space.

Algorithm #12. We modify Algorithm #4 in Section 2.1 in two ways. First, explicitly maintain a tree representing the search space that is traversed. The result is a lexicographic tree of all length-l strings that are in the d-Hamming neighborhood of a substring from each string in $\mathcal{F}$. Next, modify the pruning policy so that pruning occurs only when *all* frontiers are empty. Mark each leaf of the new lexicographic tree with the indices of the members of $\mathcal{F}$ having a substring in the d-Hamming neighborhood of the leaf. The tree that is constructed

Table 10: A Summary of the pAS Algorithms Derived in this Paper.

Alg #	Running Time	Space
10	$O(nmlp\|\Sigma\|^{lp})$	$O(nm)$
11	$O(nml + m\|\Sigma\|^{d+l}l^d + mlp\|\Sigma\|^{lp})$	$O(m\|\Sigma\|^l)$
12	$O(nm^2\|\Sigma\|^d l^{d+1} + \min(nm\|\Sigma\|^d l^{d+1}, 2^m)\, 2^m)$	$O(nm^2\|\Sigma\|^d l^{d+1} + \min(nm\|\Sigma\|^d l^{d+1}, 2^m)\, 2^m)$

encodes all strings in the d-Hamming neighborhood of a substring of any $S_i \in \mathcal{F}$. We also have a set of indices for each leaf α to indicate which $S_i \in \mathcal{F}$ has a substring in the d-Hamming neighborhood of the path label of α. Once this tree has been constructed, we use it in place of the generalized suffix tree and proceed with Algorithm #6 of Section 3.1.

The search space has grown from $O(n|\Sigma|^d l^{d+1})$ to $O(nm|\Sigma|^d l^{d+1})$ due to the fact that we now must encode the *union* of m d-Hamming neighborhoods (instead of their intersection). Since we need an m-bit vector at each leaf, the time complexity for constructing the lexicographic tree is $O(nm^2|\Sigma|^d l^{d+1})$. The space complexity has increased to equal the time complexity for this stage in the algorithm, since now we must explicitly store the lexicographic tree, where before we stored only a single path from root to leaf.

For the second stage in the algorithm, the complexity becomes $O(\min(2^m, nm|\Sigma|^d l^{d+1})\, 2^m)$. Therefore, the total time (and space) required by the algorithm is

$$O(nm^2|\Sigma|^d l^{d+1} + \min(nm|\Sigma|^d l^{d+1}, 2^m)\, 2^m).$$

The advantage of this algorithm is that exponential dependence on m can be separated from the other variables.

4.2 Parameterized Complexity of pAS

All known fixed parameter tractability results can be established through inheritance from the variant $p\textbf{AS}(\Sigma, p)$.

Theorem 4.1 *Fixed Parameter Tractability of* p-**APPROXIMATE SUBSTRING**: $p\textbf{AS}(\Sigma, l)$ *is in FPT.*

Proof. By observing that p is bounded from above by $|\Sigma|^l$, we have a bound of $O(nmlp|\Sigma|^{l\,|\Sigma|^l})$ on the total time taken by Algorithm #10 in Section 4.1. ∎

All known fixed parameter tractability results for pAS are presented in Table 11. The entries with m and n fixed are obtained by inheritance by observing that $l \leq n$ and $|\Sigma| \leq nm$.

The following reductions relate 1AS and pES to the more general problem of pAS. These allow us to partially map the hardness of pAS by using the transitivity of the $\leq_m$ relation and "lifting" results derived in Sections 2 and 3.

Table 11: Known Fixed Parameter Tractable Aspects of pAS

Parameter	–	Σ	m	m, Σ
–				
d				
l		FPT		FPT
l, d		FPT		FPT
n		FPT	FPT	FPT
n, d		FPT	FPT	FPT
n, l		FPT	FPT	FPT
n, l, d		FPT	FPT	FPT
p				
p, d				
p, l		FPT		FPT
p, l, d		FPT		FPT
p, n		FPT	FPT	FPT
p, n, d		FPT	FPT	FPT
p, n, l		FPT	FPT	FPT
p, n, l, d		FPT	FPT	FPT

Lemma 4.1 1AS $\leq_m$ pAS *such that* $p = 1$.

Proof. Given an instance $\langle \Sigma, S, l, d \rangle$ of 1AS, construct the following instance $\langle \Sigma', S', p, l', d' \rangle$ of pAS: Let $\Sigma' = \Sigma$, $S' = S$, $p = 1$, $l' = l$, and $d' = d$. ∎

Lemma 4.2 pES $\leq_m$ pAS *such that* $d = 0$.

Proof. Given an instance $\langle \Sigma, S, p, l \rangle$ of pES, construct the following instance $\langle \Sigma', S', p', l', d \rangle$ of pAS: Let $\Sigma' = \Sigma$, $S' = S$, $p' = p$, $l' = l$, and $d = 0$. ∎

Theorem 4.2 *Hardness of* p-APPROXIMATE SUBSTRING*:*

1. $p\text{AS}(m, p, l, d)$ *is* $W[1]$*-hard.*
2. $p\text{AS}(p, l, d)$ *is* $W[2]$*-hard.*
3. $p\text{AS}(\Sigma, p)$ *is not in* XP *unless* $P = NP$.
4. $p\text{AS}(n, l, d)$ *is not in* XP *unless* $P = NP$.

Proof (1). This follows from Part 1 of Theorem 2.3, and the reduction in Lemma 4.1 in which $p = 1$.

Table 12: The Parameterized Complexity of pAS

Parameter	–	Σ	m	m, Σ
–	NP-Complete	$\notin XP$	$W[1]$-Hard	
d	$\notin XP$		$W[1]$-Hard	
l	$\notin XP$	FPT	$W[1]$-Hard	FPT
l, d	$\notin XP$	FPT	$W[1]$-Hard	FPT
n	$\notin XP$	FPT	FPT	FPT
n, d	$\notin XP$	FPT	FPT	FPT
n, l	$\notin XP$	FPT	FPT	FPT
n, l, d	$\notin XP$	FPT	FPT	FPT
p	$\notin XP$	$\notin XP$	$W[1]$-Hard	
p, d	$W[2]$-Hard		$W[1]$-Hard	
p, l	$W[2]$-Hard	FPT	$W[1]$-Hard	FPT
p, l, d	$W[2]$-Hard	FPT	$W[1]$-Hard	FPT
p, n		FPT	FPT	FPT
p, n, d		FPT	FPT	FPT
p, n, l		FPT	FPT	FPT
p, n, l, d		FPT	FPT	FPT

Proof (2). Follows from Part 1 of Theorem 3.2, and the reduction in Lemma 4.2 from pES to pAS in which $d = 0$. The result also follows from Part 2 of Theorem 2.3, and the reduction in Lemma 4.1 in which $p = 1$.

Proof (3). This follows from Lemma 1.2, and the reduction in Lemma 4.2.

Proof (4). This follows from Lemma 1.2, and the reductions in Lemma 3.1 and Lemma 4.2. ∎

Corollary 4.3 *There is no polynomial time approximation algorithm relative to p for pAS when $|\Sigma| \geq 2$ with a logarithmic factor ($c \log p$ for any constant c) unless $P = NP$.*

Proof. Follows from Corollary 3.4 and the L-reduction in in Lemma 4.2. ∎

5 Conclusions and Future Research

These results show that many of the problems examined here become fixed parameter-tractable with between one and four of the aspects considered here. This is only the beginning of algorithmic work on this problem. Our experience working on DNA primer design problems [5] has shown that even for variants composed of many aspects that "inherit" FPT algorithms based on subsets of these aspects, it is still very useful to do further algorithm development in order to produce the best useable algorithms. Different applications can

lead to different restrictions on the parameters, such as the primer design and motif finding challenge problems. These applications benefit from algorithms whose parameterized time and space useage are most suited to the parameter restrictions inherent in the problem. Probe and primer design, which use larger lengths and smaller distances, can have a more suitable algorithm than that used for general motif finding, which has smaller lengths but larger distances. With respect to the parameterized complexity results presented in Tables 6, 9 and 12, some observations are worth noting:

- Among aspects of 1**AS**, the best candidate for membership in FPT is 1**AS**(m, Σ, d).
- It is difficult to imagine 1**AS**(m, Σ) residing in FPT. However, it has yet to be shown hard for any class in the W-hierarchy.
- Our analysis has not revealed a difference in complexity between the aspects of 1**AS** in the first row and those in the second row of Table 6. What is the effect of fixing d alone?
- Considering the fixed parameter tractability of 1**AS**(n) and p**ES**(p, n), it is plausible that p**AS**(p, n) is also in FPT. Is this the case?
- The bottom half of Table 12 is quite similar to Table 6. None of our results suggest a difference in complexity between p**AS**(p) and 1**AS**.

In addition to determining the parameterized complexity of p**ES**, 1**AS**, and p**AS** relative to the few remaining "open" parameters, there are several other promising directions for future research:

- What is the parameterized complexity of the non-trivial versions of p**AS** relative to the third dimension mentioned in the Introduction, *i.e.*, 1**AS** $(l = n)$ (**CLOSEST STRING**) and p**AS** $(l = n)$? Such results might in turn shed further light on the parameterized complexity of 1**AS** and p**AS**.
- What types of polynomial-time approximation algorithms exist for p**ES** and p**AS**? Given the exponential-time approximation scheme for p**AS** when $l = n$ and $|\Sigma| = 2$ described in [8], it would also be of interest to investigate the existence of fixed parameter approximation schemes for 1**AS**, p**ES**, and p**AS** [3].

Such research would be a useful prelude to examinations of the parameterized complexity of various string median problems under Hamming and edit distance [11, 13].

References

[1] G. Ausiello, A. D'Atri, and M. Protasi. 1980. Structure Preserving Reductions Among Convex Optimization Problems. *Journal of Computer and System Sciences*, 21, 136–153.

[2] M. Blanchette, B. Schwikowski, and M. Tompa. 2000. An Exact Algorithm to Identify Motifs in Orthologous Sequences from Multiple Species. In *ISMB 2000*. AAAI Press. 37–45.

[3] M. Cesati and L. Trevisan. 1997. On the Efficiency of Polynomial Time Approximation Schemes. *Information Processing Letters*, 64(4), 165–171.

[4] R.G. Downey and M.R. Fellows. 1999. *Parameterized Complexity*. Springer-Verlag; Berlin.

[5] P. Evans and T. Wareham. 2001. Practical Algorithms for Universal DNA Primer Design: An Exercise in Algorithm Engineering (Poster Abstract). In N. El-Mabrouk, T. Lengauer, and D. Sankoff (eds.) *Currents in Computational Molecular Biology 2001*. Les Publications CRM; Montreal, PQ. 25–26.

[6] M. Frances and A. Litman. 1997. On Covering Problems of Codes. *Theory of Computing Systems*, 30, 113–119.

[7] M.R. Garey and D.S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company; San Francisco.

[8] L. Gąsieniec, J. Jansson, and A. Lingas. 2000. Approximation Algorithms for Hamming Clustering Problems. In R. Giancarlo and D. Sankoff (eds.) *CPM 2000*. Lecture Notes in Computer Science no. 1848. Springer-Verlag; Berlin. 108–118.

[9] J. Gramm, R. Niedermeier, and P. Rossmanith. 2001. Exact Solutions for Closest String and Related Problems. To appear, *ISAAC'01*.

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

[11] C. de la Higuera and F. Casacuberta. 2000. Topology of strings: Median string is NP-complete. *Theoretical Computer Science*, 230, 39–48.

[12] J.K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. 1999. Distinguishing String Selection Problems. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM Press; New York. 633–642.

[13] M. Li, B. Ma, and L. Wang. 1999. Finding Similar Regions in Many Sequences. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*. ACM Press; New York. 473–482.

[14] B. Ma. 2000. A Polynomial Time Approximation Scheme for the Closest Substring Problem. In R. Giancarlo and D. Sankoff (eds.) *CPM 2000*. Lecture Notes in Computer Science no. 1848. Springer-Verlag; Berlin. 99–107.

[15] C. Papadimitriou and K. Steiglitz. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall; Englewood Cliffs, NJ.

[16] C. Papadimitriou and M. Yannakakis. 1991. Optimization, Approximation, and Complexity Classes. *Journal of Computer and System Sciences*, 43, 425–440.

[17] P.A. Pevzner and S.-H. Sze. 2000. Combinatorial Approaches to Finding Subtle Signals in DNA Sequences. In *ISMB 2000*. AAAI Press. 269–278.

[18] R. Raz and S. Safra. 1997. A Sub-Constant Error-Probability Low-Degree-Test and a Sub-Constant Error-Probability PCP Characterization of NP. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*. ACM Press; New York. 475–484.

[19] M.-F. Sagot. 1998. Spelling approximate repeated or common motifs using a suffix tree. In V.L. Lucchesi and A. Moura (eds.) *LATIN'98*. Lecture Notes in Computer Science no. 1380. Springer-Verlag; Berlin. 111–127.

[20] M.-F. Sagot, V. Escalier, A. Viari, and H. Soldana. 1995. Searching for repeated words in a text allowing for mismatches and gaps. In *Proceedings of the Second South American Workshop on String Processing*. University of Chile. 87–100.

[21] E. Ukkonen. 1993. Approximate String Matching over Suffix Trees. In *CPM 1993*. Lecture Notes in Computer Science no. 684. Springer-Verlag; Berlin. 228–242.

[22] E. Ukkonen. 1995. On-line construction of suffix trees. *Algorithmica*, 14, 249–260.

[23] H.T. Wareham. 1999. *Systematic Parameterized Complexity Analysis in Computational Phonology*. Ph.D. thesis, Department of Computer Science, University of Victoria.

[24] M.S. Waterman, R. Arratia, and D.J. Galas. 1984. Pattern recognition in several sequences: consensus and alignment. *Bulletin of Mathematical Biology*, 46, 515–527.