Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®
Theoretical Computer Science 306 (2003) 407–430
Theoretical Computer Science
www.elsevier.com/locate/tcs

On the complexity of finding common approximate substrings

Patricia A. Evans[a,*,1], Andrew D. Smith[a,2], H. Todd Wareham[b,3]

[a] *Faculty of Computer Science, University of New Brunswick, P.O Box 4400, Fredericton, NB, Canada, E3B 5A3*

[b] *Department of Computer Science, Memorial University of Newfoundland, St.John's, NF, Canada, A1B 3X5*

Received 2 July 2002; received in revised form 11 March 2003; accepted 12 May 2003
Communicated by A. Apostolico

Abstract

Problems associated with finding strings that are within a specified Hamming distance of a given set of strings occur in several disciplines. In this paper, we use techniques from parameterized complexity to assess non-polynomial time algorithmic options and complexity for the **COMMON APPROXIMATE SUBSTRING** (**CAS**) problem. Our analyses indicate under which parameter restrictions useful algorithms are possible, and include both class membership and parameterized reductions to prove class hardness. In order to achieve fixed-parameter tractability, either a fixed string length or both a fixed size alphabet and fixed substring length are sufficient. Fixing either the string length or the alphabet size and Hamming distance is shown to be necessary, unless $W[1] = FPT$. An assortment of parameterized class membership and hardness results cover all other parameterized variants, showing in particular the effect of fixing the number of strings.

Keywords: Parameterized complexity; Pattern matching; Common approximate substring

* Corresponding author.
E-mail addresses: pevans@unb.ca (P.A. Evans), p7ka@unb.ca (A.D. Smith), harold@cs.mun.ca (H.T. Wareham).

[1] Supported by NSERC operating grant 204923.
[2] Supported by NSERC PGSA 232124.
[3] Supported by NSERC operating grant 228104.

0304-3975/03/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0304-3975(03)00320-7

1. Introduction

Finding strings that are approximately present in each of a given set of strings is a problem common to many areas of application, including computational biology. What "approximately present" means can vary, including the existence of different and arbitrarily-sized gaps in instances of the found string. In this paper, we restrict the approximate presence of a string $\mathscr{C}$ in a given string S to mean the existence of a substring of S that is within a specified Hamming distance of $\mathscr{C}$. That substring of S is considered to be an (*approximate*) *instance* of $\mathscr{C}$, and the string $\mathscr{C}$ is considered to be an *approximate substring* of S. If $\mathscr{C}$ is an approximate substring of each string in a set $\mathscr{F}$ of strings, then it is a *common approximate substring* for $\mathscr{F}$.

Formally, 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:

COMMON APPROXIMATE SUBSTRING (CAS)

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

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

We call $\mathscr{C}$ the *center string* for $\mathscr{F}$. When comparing strings, we use the notation $d_H(a,b)$ to denote the Hamming distance between strings a and b. When $|a|<|b|$, $d_H(a,b)=\min_{b'\in B} d_H(a,b')$, where B is the set of all substrings of b having length $|a|$. When $l=n$, CAS is known as COVERING RADIUS [7] and CLOSEST STRING [11] and has been investigated in the context of coding theory. When $l\leqslant n$, CAS is known as CLOSEST SUBSTRING [11,12] 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 by results proved independently in [7,11] for CLOSEST STRING.

Though several CAS variants are admittedly trivial or are known to be solvable in low-order polynomial time (when $l<n$ and $d=0$, i.e., LONGEST COMMON SUBSTRING), the *NP*-hardness of the problems mentioned above suggest that the remaining CAS variants are much more difficult. Polynomial-time approximation algorithms seem a natural first choice for solving these problems in practice. Several such algorithms for CLOSEST SUBSTRING that give solutions within a multiplicative factor of 2 of the optimal value of d are known [11,12], and a polynomial-time approximation scheme (PTAS) has also recently been developed [13]. Unfortunately, the high degree of the polynomial in the running time of the PTAS renders it of theoretical interest only. There are, however, other less traditional routes to practical algorithms. The dependence of the *NP*-hardness results [7,11] on the hardness of CLOSEST STRING, a special case where $l=n$, indicates that this problem is worth investigating using fixed-parameter techniques, e.g. for $l\ll n$. The CLOSEST STRING problem itself 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].

In this paper, we use techniques developed within the theory of parameterized computational complexity [3] to systematically examine the possible types of useful non-polynomial time algorithms for CAS. 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 [17].

Fixed-parameter techniques have previously been used to explore variants of finding common subsequences. A *subsequence* of a string s is a string composed of symbols from s which may be separated by arbitrarily-sized gaps in s. Substrings, on the other hand, must be composed of symbols occurring consecutively, without gaps; so all substrings are subsequences, but some subsequences are not also substrings. For example, given the string $s = abcabdacb$, the strings $cabd$, abc, and dac are all both substrings and subsequences of s, while bcb and cda are only subsequences of s. The problem of finding a common subsequence of length l for m sequences, each of length n, is NP-hard [14], and also hard to approximate [10]. Its variants have been examined systematically using parameterized complexity; it is $W[t]$-hard for all t when parameterized by m and $|\Sigma|$, $W[2]$-hard when parameterized by l, and $W[1]$-complete when parameterized by m and l together [2].

In the case of CAS, we show that useful algorithms are possible if the substring length l is fixed together with the alphabet size $|\Sigma|$; more effective algorithms can result if additional parameters are also restricted.

1.1. Parameterized complexity analysis

According to the theory of *NP*-completeness [8], 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, e.g., $|\Sigma|$, d, l, and m in the case of CAS. When one or more aspects of a problem are "fixed" in this manner, we indicate this by placing these aspects in parentheses after the problem name, i.e., CAS with fixed alphabet size and fixed number of strings is written as CAS$(m, |\Sigma|)$.

The theory of parameterized computational complexity [3] provides explicit mechanisms for analyzing the effects of individual aspects on problem complexity. Within this theory, a decision problem in which a set of aspects is fixed is called a *parameterized problem* and the set of fixed aspects is referred to as that problem's *parameter*. Given these notions, the most basic definition within this theory is that of a tractable parameterized problem.

Definition 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 $f : \Sigma^{+} \mapsto \mathbb{N}$ is an arbitrary function and α is a constant independent of x and p.

Definition 2. A parameterized problem Π belongs to the class FPT if Π is fixed-parameter tractable.

There are a variety of techniques for deriving FPT algorithms for parameterized problems (see [3] and references therein). One can establish that a parameterized problem Π is not fixed-parameter tractable by using a parametric reduction[4] to show that Π is hard for any of the classes of the W-*hierarchy* $= \{W[1], W[2], \ldots, W[P], XP\}$. These classes are the parameterized analogs of the class NP in the theory of NP-completeness. They are, for the most part, based on a series of successively more powerful solution-checking circuits in which solutions are encoded as input vectors to these circuits and parameters are encoded in the weights of these input vectors.

Definition 3. A *boolean circuit* α_n with input $x = x_1 x_2 \cdots x_n$ of length n is a directed acyclic graph. The nodes of fan-in 0 are called input nodes and are labeled from the set $\{0, 1, x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\}$. The nodes of fan-in greater than 0 are called gates and are labeled either AND or OR. 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. A *truth assignment* is a binary vector $x = x_1 \ldots x_n \in \{0, 1\}^n$, where value assigned to the ith input node is 1 if and only if $x_i = 1$.

Definition 4. A gate is said to have *unbounded fanin* if the number of inputs to that gate exceeds some constant bound. The *weft* of a decision circuit is the maximum number of gates with unbounded fanin on any path from the input variables to the output node.

WEIGHTED WEFT t DEPTH h CIRCUIT SATISFIABILITY (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 truth assignment? A weight k satisfying truth assignment is a truth assignment to the inputs of C that both satisfies C, and assigns the value 1 to exactly k input nodes.

Definition 5. Let t be a positive constant. A parameterized problem Π belongs to the class $W[t]$ if Π parametrically reduces to WCS$_{t,h}$ for some h.

Definition 6. A parameterized problem Π belongs to the class $W[P]$ if it parametrically reduces to WCS$_{t,h}$. Note that t and h are not explicitly restricted, but required by the definition of a parameterized reduction to be polynomial functions of $|\Pi|$.

[4] Given parameterized problems $\Pi(p)$ and $\Pi'(p')$, $\Pi(p)$ *parametrically (many-one) 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 $f(p)|x|^{\alpha}$ time for an arbitrary function f independent of x 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 $x' \in \Pi'(p')$.

Definition 7. A parameterized problem Π belongs to the class XP if there exists an algorithm A to determine if instance (x, y) is in Π in time $f(y) \cdot |x|^{g(y)}$, where $f : \Sigma^+ \mapsto \mathbb{N}$ and $g : \Sigma^+ \mapsto \mathbb{N}$ are arbitrary functions.

These classes have the following inclusions:

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

If a parameterized problem can be shown to be $\mathscr{C}$-hard for any class $\mathscr{C}$ 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 $FPT \neq \mathscr{C}$.

The following lemma is used in the analyses given in later sections of this paper; it exploits *NP*-hardness results relative to constant-valued aspects to conjecture a weak form of fixed-parameter intractability for certain parameterized problem variants.

Lemma 8 (Wareham [17, 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.

The nature of the COMMON APPROXIMATE SUBSTRING PROBLEM is that relations exist between many of the parameters. The following lemma is used to extend results by simply noting these relations.

Lemma 9. *Let $\mathscr{C}$ be a class in the W-hierarchy and let Π be a parameterized problem with parameters k_1 and k_2. If there exists some function f, such that $k_1 \leqslant f(k_2)$ for all instances of Π,*

(1) *If $\Pi(k_1) \in \mathscr{C}$, then $\Pi(k_2) \in \mathscr{C}$,*
(2) *If $\Pi(k_2)$ is $\mathscr{C}$-hard, then $\Pi(k_1)$ is $\mathscr{C}$-hard.*

Section 2 describes CAS problem variants that are fixed-parameter tractable. For most variants not known to be in FPT, Section 3 gives hardness results for various classes of the W-hierarchy. In Section 4 we describe circuits used to establish membership of variants in the W-hierarchy. All parameterized analyses in these sections were done relative to the following aspects:

- the size of the alphabet ($|\Sigma|$),
- the number of strings in $\mathscr{F}(m)$,
- the length of the strings in $\mathscr{F}(n)$,
- the length of the requested substrings (l), and
- the Hamming distance threshold (d).

All results derived within or implied by these analyses are summarized in Table 1. Although n is usually considered to be large with respect to other problem aspects, we include it in our analyses for the sake of completeness, and to provide an algorithm for a large set of short strings over an unrestricted alphabet. Section 5 discusses implications of this work for related Common Subsequence problems. Section 6 gives some promising directions for future research.

Table 1
Parameterized complexity of **COMMON APPROXIMATE SUBSTRING**

Parameter	—	Σ	m	m, Σ
–	*NP*-Complete	$\notin XP$	$W[2]$-Hard	$W[2]$-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*
n	*FPT*	*FPT*	*FPT*	*FPT*

2. Fixed-parameter tractability results

Fixed-parameter tractable algorithms for **COMMON APPROXIMATE SUBSTRING** are of particular interest in computational biology for finding short segments that approximately occur in entire families of DNA or RNA sequences. These segments 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 general sequence family motifs (see [5,11] and references theirin). Typically these examples require only small parameter values. For example, instances of **CAS** 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., $|\Sigma| = 4$, $d \leqslant 3$, and $l \leqslant 25$ [5]. Similarly, a general search for common sequence motifs has produced a challenge problem in the computational biology community—namely, **CAS** when $n = 600$, $m \geqslant 15$, $|\Sigma| = 4$, $l = 15$, and $d = 4$ [16].

Given numerical dependencies among problem aspects, with $d \leqslant l \leqslant n$ and $|\Sigma| \leqslant mn$, the following two algorithms are sufficient to establish the fixed-parameter tractability of all tractable parameterized variants discussed in this paper. Note that algorithms that restrict dependent aspects independently can attain lower complexities, and are therefore superior for some applications [4].

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 $\mathscr{F}$. 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 and $O(mn)$ space. This is essentially the first algorithm given in [18]. The advantage of this approach is that the total time and space required is a linear function of the input size.

Algorithm 2. 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 $\mathscr{F}$. Let $\mathscr{C}$ be a length l string containing at most d occurrences of the blocking character x. Let s be a length l substring of string $S \in \mathscr{F}$. A substitution of $\mathscr{C}$ under s is a set of modifications to $\mathscr{C}$ that replaces a subset of the occurrences of character x in $\mathscr{C}$ with the characters appearing in corresponding positions of s. A *minimal matching* substitution is a substitution that results in $d_H(\mathscr{C}, s) \leqslant d$, with the additional property that no substitution replacing *a subset of those same positions* results in $d_H(\mathscr{C}, s) \leqslant d$ (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 $S \in \mathscr{F}$ is removed from $\mathscr{F}$. For each length l substring s of S, let $\mathscr{C} = s$, and for each size d set of positions in $\mathscr{C}$, change the characters at those positions to the blocking character x. The second stage of the algorithm proceeds recursively, developing a center $\mathscr{C}$ by substituting characters of Σ back into positions occupied by the blocking character. For each recursive call, a string S' is arbitrarily removed from $\mathscr{F}$. If the center $\mathscr{C}$ 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 $\mathscr{C}$ in more than d positions, alternative centers are produced by modifying $\mathscr{C}$. For each substring $s \in S'$, such that $d_H(s,\mathscr{C}) \leqslant 2d$, a set M of alternative centers is obtained by making a minimal matching substitution to $\mathscr{C}$ under s. For each $\mathscr{C}'$ in M, a recursive call is made, with $\mathscr{C}'$ passed as the center to further develop. If the set $\mathscr{F}$ becomes empty, then the center currently being developed is valid for the *original* set of strings $\mathscr{F}$ (i.e., before any strings were removed), and the algorithm returns that center.

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

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

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

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

DevelopCenter($\mathscr{F}, \mathscr{C}$)

1. **if** $\mathscr{F} = \emptyset$ **return** $\mathscr{C}$
2. **if** $\mathscr{C} = \lambda$
3. Let S be an arbitrary string in $\mathscr{F}$.
4. **for each** length l substring s of S
5. $\mathscr{C} \leftarrow s$

```
6.          for each B ⊆ {1,...,l}, such that |B| = d
7.              ∀ positions p ∈ B, substitute 𝒞[p] ← x
8.              DevelopCenter 𝓕\{S}, 𝒞
10.   if 𝒞 ≠ λ
11.       Let S be an arbitrary string in 𝓕.
12.       branch ← true
13.       for each length l substring s of S
14.           if d_H(s, 𝒞) ⩽ d then branch ← false
15.       if branch = false then DevelopCenter(𝓕\{S}, 𝒞)
16.       if branch = true
17.           for each length l substring s of S such that d_H(s, 𝒞) ⩽ 2d
18.               Let M be the set mm(𝒞, s).
19.               for each 𝒞' ∈ M
20.                   DevelopCenter(𝓕\{S}, 𝒞')
```

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 maximum number of leaves in the search space is $n\binom{l}{d}\left(n\binom{d}{d/2}\right)^d$ and the length of any path from root to leaf is m. At each node in the search tree, $O(n)$ time is required to determine if a branch is necessary. Hence, the time complexity of the algorithm is bounded by $O(n^2m\binom{l}{d}\left(\binom{d}{d/2}n\right)^d)$. Of note is the absence of any exponential factor involving alphabet size in this time complexity expression.

Theorem 10. *Fixed-parameter tractability of* COMMON APPROXIMATE SUBSTRING:
(1) CAS$(|\Sigma|, l) \in FPT$.
(2) CAS$(n) \in FPT$.

Proof. (1): Follows from Algorithm 1 above, which solves CAS in $O(l|\Sigma|^l nm) = O(f(|\Sigma|, l)nm)$ time.

(2): Follows from the fact that $d \leqslant l \leqslant n$ for CAS and Algorithm 2 above, which solves CAS in $O(l^d d^{d^2} n^{d+2} m) = O(f(n)m)$ time. □

The results in this theorem anchor all fixed-parameter tractability results for CAS presented in this paper—in particular, all fixed-parameter tractability results in the bottom row of Table 1 arise from Part (2) (in which n is fixed) and the remaining fixed-parameter tractability results in this table arise from Part (1) (in which $|\Sigma|$ and l are fixed). The problem variants associated with most of the remaining sets of aspects are intractable in the parameterized setting, as proved in the following section.

3. Parameterized Hardness Results

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

CLIQUE [8, 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 [3, p. 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 [8, Problem SP5]

Instance: A set $\mathscr{B}$ of elements, a family of sets $\mathscr{L}$ such that $\mathscr{L}_i \subseteq \mathscr{B}, (1 \leqslant i \leqslant |\mathscr{L}|)$ and a positive integer k.
Parameter: A positive integer k.
Question: Is there a size k subset $R \subseteq \mathscr{L}$ such that $\bigcup_{R_j \in R} R_j = \mathscr{B}$?

3.1. The $W[1]$-hardness of CAS(m,l,d)

To show $W[1]$-hardness for CAS(m,l,d), we give a parameterized reduction from the $W[1]$-complete problem CLIQUE [3]. Note that versions of this reduction were developed independently in [4,6]. Let $G=(V,E)$ be a graph for which we wish to determine whether G has a k-clique. We construct a family $\mathscr{F}$ 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 $V=\{1,\ldots,|V|\}$.

Target parameters: The number of strings in $\mathscr{F}$ is $m=f_1(k)=\binom{k}{2}$. The length of center $\mathscr{C}$ 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 $\mathscr{F}$ (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):

$$\Sigma_1 = \{1,\ldots,|V|\},$$
$$\Sigma_2 = \{\text{Set of characters occurring uniquely in } \mathscr{F}\},$$
$$\Sigma_3 = \{A,B\}.$$

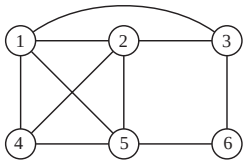

Fig. 1. Graph 1.

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:

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

Separators:

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

For the Edge Selectors, the index pair (i,j) specifies a *clique edge*, and (p,q) specifies an edge, from the underlying graph, that may form the clique edge (i,j) in a size k clique. The characters p and q in the specification of Edge Selectors are from Σ_1.

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

$$\mathscr{F} = \{S_{ij} : 1 \leqslant i < j \leqslant k\}.$$

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

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

An example of the reduction for the graph in Fig. 1 and a desired clique size 4 can be seen in Fig. 2. Any center for $\mathscr{F}$ has the property that all positions other than the terminal positions are occupied by vertex characters in ascending order.

The proof of correctness of this reduction exploits some interesting properties of $\mathscr{F}$. These properties are examined with the aid of the following definitions and conventions. 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 $\mathscr{V}_i$, is defined as the set

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

S_{12}: $A12uuBu^6A13uuBu^6A14uuBu^6A23uuBu^6A24uuBu^6$
S_{13}: $A1u3uBu^6A1u4uBu^6A1u5uBu^6A2u4uBu^6A2u5uBu^6$
S_{14}: $A1uu4Bu^6A1uu5Bu^6A2uu5Bu^6A3uu6Bu^6$
S_{23}: $Au23uBu^6Au24uBu^6Au25uBu^6Au45uBu^6$
S_{24}: $Au2u4Bu^6Au2u5Bu^6Au3u6Bu^6$
S_{34}: $Auu36Bu^6Auu45Bu^6Auu56Bu^6$

Fig. 2. **CAS**(m, l, d) representation for Graph 1 (desired clique size $k = 4$).

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

Lemma 11. *Let $\mathscr{C}$ be a center for $\mathscr{F}$. The following are true*:
(1) *$\mathscr{C}$ begins with character A and ends with character B.*
(2) *No position in $\mathscr{C}$ is occupied by a character from Σ_2.*
(3) *If $\mathscr{I}$ is an instance of $\mathscr{C}$, then $\mathscr{I}$ is in-phase.*
(4) *The $k-1$ instances from any vertex group are sufficient to completely determine $\mathscr{C}$.*

Proof.

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

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

(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.

(4) Suppose $\mathscr{C}$ has been partially determined by instances from $\mathscr{V}_z' \subset \mathscr{V}_z$, for vertex position z. Consider instance $\mathscr{I}$ from $S_{z,x} \in \mathscr{V}_z \backslash \mathscr{V}_z'$. By Parts (2) and (3), $\mathscr{I}$ must match the alignment positions, and positions z and x. Since $\mathscr{I}$ is the only member of $\mathscr{V}_z$ that can determine a non-unique character at position x, that position has not yet been determined. In order for $\mathscr{I}$ to match 4 positions of $\mathscr{C}$, $\mathscr{I}$ must determine position x in $\mathscr{C}$. 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 12. CLIQUE *parametrically reduces to* CAS(m,l,d).

Proof. The construction described above 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. The resulting string is clearly a center for $\mathscr{F}$, with instances being the Edge Selectors corresponding to the edges in the k-clique from G. Conversely, suppose there is a center $\mathscr{C}$ for a set of strings $\mathscr{F}$, constructed from a graph G as per the reduction described above. By Lemma 11, all instances match $\mathscr{C}$ at exactly 2 positions occupied by characters from Σ_1. By the reduction, there are $\binom{k}{2}$ edges in G incident on a total of k vertices. Hence there is a clique of size k in G. □

Theorem 13. CAS(m,l,d) *is* $W[1]$*-hard.*

Proof. Follows from Lemma 12 and the W[1]-hardness of CLIQUE [3]. □

3.2. The $W[2]$-hardness of CAS(l,d)

To show $W[2]$-hardness for CAS(l,d), we give a parameterized reduction from the $W[2]$-complete problem DOMINATING CLIQUE [3]. 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 $\mathscr{F}$ 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 $V=\{1,\ldots,x\}$. The alphabet and substring gadgets are exactly the same as for the reduction in Section 3.1.

The target parameters: The number of strings in $\mathscr{F}$ 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 the reduction in Section 3.1.

The reduction: The strings form two groups $\mathscr{F}=\mathscr{F}_{G_E}\cup\mathscr{F}_{G_V}$ having distinct roles. The $\binom{k}{2}$ strings in $\mathscr{F}_{G_E}$ 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 $\mathscr{F}_{G_V}$ are responsible for verifying that any center determined by instances from $\mathscr{F}_{G_E}$ corresponds not only to a k-clique, but to a dominating set as well:

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

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

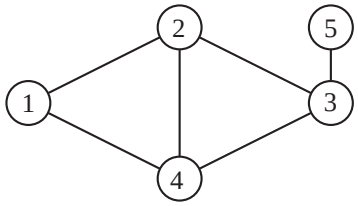

Fig. 3. Graph 2.

S_{12}: $A12uBu^5A14uBu^5A23uBu^5A24uBu^5A34uBu^5$
S_{13}: $A1u4Bu^5A2u4Bu^5A3u5Bu^5$
S_{23}: $Au23Bu^5Au24Bu^5Au34Bu^5Au35Bu^5$

S_{V_1}: $A12uBu^5A14uBu^5A23uBu^5A24uBu^5A34uBu^5$
$A1u4Bu^5A2u4Bu^5Au23Bu^5Au24Bu^5Au34Bu^5$
S_{V_2}: $A12uBu^5A14uBu^5A23uBu^5A24uBu^5A34uBu^5$
$A1u4Bu^5A2u4Bu^5A3u5Bu^5Au23Bu^5Au24Bu^5$
$Au34Bu^5Au35Bu^5$
S_{V_3}: $A12uBu^5A14uBu^5A23uBu^5A24uBu^5A34uBu^5$
$A1u4Bu^5A2u4Bu^5A3u5Bu^5Au23Bu^5Au24Bu^5$
$Au34Bu^5Au35Bu^5$
S_{V_4}: $A12uBu^5A14uBu^5A23uBu^5A24uBu^5A34uBu^5$
$A1u4Bu^5A2u4Bu^5A3u5Bu^5Au23Bu^5Au24Bu^5$
$Au34Bu^5Au35Bu^5$
S_{V_5}: $A23uBu^5A34uBu^5A3u5Bu^5Au23Bu^5Au34Bu^5$
$Au35Bu^5$

Fig. 4. CAS(l,d) representation for Graph 2 (desired dominating clique size $k=3$).

neighbors of x):

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

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

Lemma 14. DOMINATING CLIQUE *parametrically reduces to* CAS(l,d).

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

in Lemma 12, a center for $\mathscr{F}_{G_E}$ 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 may serve as an instance for the center. Therefore a dominating k-clique in G implies a center for $\mathscr{F}$.

Suppose there is a center $\mathscr{C}$ for $\mathscr{F}$. By the reduction in Section 3.1, $\mathscr{C}$ specifies a k-clique in G. Since $\mathscr{C}$ has instances in all strings from $\mathscr{F}_{G_V}$, and those instances match $\mathscr{C}$ at 2 positions occupied by characters of Σ_1, the corresponding vertices in V are adjacent to a vertex in the clique specified by $\mathscr{C}$. Hence there is a size k dominating clique in G. □

Theorem 15. CAS(l,d) *is* W[2]*-hard.*

Proof. Follows from Lemma 14 and the $W[2]$-hardness of DOMINATING CLIQUE [3]. □

3.3. The $W[2]$-hardness of CAS$(m,|\Sigma|)$

To show $W[2]$-hardness for CAS$(m,|\Sigma|)$, we give a parameterized reduction from the $W[2]$-complete problem SET COVER [3]. This result both strengthens and complements the $W[1]$-hardness result given in [6] for CAS$(m,|\Sigma|)$ when $|\Sigma|=2$.

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

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

$$\Sigma_1=\{s_1,\ldots,s_k\},$$
$$\Sigma_2=\{u_{11},u_{12},u_{21},u_{22},\ldots,u_{k1},u_{k2}\}.$$

For $1\leqslant i\leqslant k$, 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:

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

Separators:

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

Subset indicators:

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

The Fillers are strings of length $(k-1)|\mathscr{B}|$ and each corresponds to some $\mathscr{L}_i \in \mathscr{L}$. The Separators are strings of length $k|\mathscr{B}|+2$. 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

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

The reduction: Each of the k sets in the solution R of I is represented by a pair of strings in $\mathscr{F}$. In particular, the instances in strings $S_{i1}, S_{i2} \in \mathscr{F}$ correspond to the ith set in R. Define

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

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

The family of strings is then $\mathscr{F} = \{S_{11}, S_{12}, S_{21}, S_{22}, \ldots, S_{k1}, S_{k2}\}$. Note that no matter what set of substrings is taken as instances 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 Figs. 5–7.

For Fig. 6, the subscripts are left out of the unique characters; these are given unique symbols in Fig. 7.

Define $\bar{d}$ as the minimum possible value of d for a set of instances. 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 instances of a center string for $\mathscr{F}$, define z_{ij} as the indicator function that has the value 1 if $S_i[j]$ is *not* the column majority character in column j of the aligned instances and the value 0 otherwise (in case there are two or more characters that may serve as column majority, one is arbitrarily selected). Given

$\mathscr{B}$: $\{1,2,3,4,5,6,7\}$
$\mathscr{L}_1$: $\{1,4,6\}$
$\mathscr{L}_2$: $\{1,2,4\}$
$\mathscr{L}_3$: $\{3,5\}$
$\mathscr{L}_4$: $\{1,2,3,7\}$
k: 3

Fig. 5. Example instance of **SET COVER**.

S_{11} : A1uu1u1u1^{14}Au^{23}A11u1uuu1^{14}Au^{23}Auu1u1uu1^{14}Au^{23}A111uuu11^{14}Au^{23}
S_{12} : 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}Auu2u2uu2^{14}Au^{23}A222uuu22^{14}Au^{23}
S_{22} : A2uu2u2u2^{14}Au^{23}A22u2uuu2^{14}Au^{23}Auu2u2uu2^{14}Au^{23}A222uuu22^{14}Au^{23}
S_{31} : A3uu3u3u3^{14}Au^{23}A33u3uuu3^{14}Au^{23}Auu3u3uu3^{14}Au^{23}A333uuu33^{14}Au^{23}
S_{32} : A3uu3u3u3^{14}Au^{23}A33u3uuu3^{14}Au^{23}Auu3u3uu3^{14}Au^{23}A333uuu33^{14}Au^{23}

Fig. 6. CAS($m, |\Sigma|$) representation for the example instance of **SET COVER** given in Fig. 5. The character u denotes a character that differs across strings. The underlined substrings are expanded and explained in Fig. 7.

$S_{11}[1]$:	A1aa1a1a11111111111111A
$S_{12}[1]$:	A1bb1b1b11111111111111A
$S_{21}[93]$:	Acc2c2cc22222222222222A
$S_{22}[93]$:	Add2d2dd22222222222222A
$S_{31}[139]$:	A333eee333333333333333A
$S_{32}[139]$:	A333fff333333333333333A
Center String:	A3331213111112222222333A

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

Fig. 7. Expanded diagrams of the substrings underlined in Fig. 6, along with an optimal center string for the instances. Underlined characters are those matching corresponding positions in the center.

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 16. $\hat{d} \leqslant \bar{d}$.

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

As a corollary to Lemma 16, the following lemma is derived by substituting the appropriate values into the formula for $\hat{d}$.

Lemma 17. *If the column majority character occurs at most twice in any column, then* $\bar{d} \geqslant l - 2l/m$.

Lemma 18. *Let* $\mathscr{F}$ *be a set of strings constructed as described in the reduction and let* $\mathscr{C}$ *be a center for* $\mathscr{F}$. *Then* $\mathscr{C}$ *must begin and end with the alignment character, and so must all instances.*

Proof. Suppose the center $\mathscr{C}$ does not begin with the alignment character; then by the separation between alignment characters in members of $\mathscr{F}$, no instance can match two alignment characters in $\mathscr{C}$. 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 17 with the substitutions $m = 2k$ and $l = k|\mathscr{B}| + 2$ to obtain $\bar{d} \geqslant (k-1)|\mathscr{B}| + 2(k-1)/k$. A symmetric argument establishes that $\mathscr{C}$ must end with the alignment character.

Suppose some instance begins or ends with a character other than the alignment character. Then that instance cannot match $\mathscr{C}$ at those positions. Again using Lemma 17, $\bar{d} \geqslant (k-1)|\mathscr{B}| + (k-1)/k$. □

Lemma 19. 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 $\mathscr{B}$, such that $|R| = k$. From R, construct a center $\mathscr{C}$ for $\mathscr{F}$ as follows. (1) The first and last positions of $\mathscr{C}$ are assigned the alignment character A. (2) The next $|\mathscr{B}|$ 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 $b \in \mathscr{B}$, choose some $\mathscr{L}_i \in R$, such that $b \in \mathscr{L}_i$, as covering b. Since R is a cover for $\mathscr{B}$, there is at least one such choice for every $b \in \mathscr{B}$. If $\mathscr{L}_i$ is chosen to cover b, then s_i is assigned to position $b+1$ in $\mathscr{C}$ (recall that the elements of $\mathscr{B}$ have been equated with the integers 1 to $|\mathscr{B}|$). (3) The remaining $(k-1)|\mathscr{B}|$ positions of $\mathscr{C}$ correspond to the Filler gadgets. For each $\mathscr{L}_i \in R$, if x_i positions ($0 \leqslant x_i \leqslant |\mathscr{B}|$) of $\mathscr{C}$ have been assigned characters corresponding to elements in $\mathscr{L}_i$, then $|\mathscr{B}| - x_i$ positions in the Filler part of $\mathscr{C}$ are assigned the character s_i. If $\mathscr{L}_j \in \mathscr{L}$ is the ith set in R, then the substring of S_{i1} (and of S_{i2}) that begins and ends with the alignment character, and contains the jth Subset Indicator, matches $\mathscr{C}$ in exactly $2 + |\mathscr{B}|$ positions. Therefore $\mathscr{C}$ is a center for $\mathscr{F}$ with distance exactly $(k-1)|\mathscr{B}|$ to any instance.

Suppose there is a $(k-1)|\mathscr{B}|$ center $\mathscr{C}$ for $\mathscr{F}$ constructed as per the reduction from an instance of SET COVER. By Lemma 17, the column majority character occurs at least twice in each column, implying all but the first and last positions of $\mathscr{C}$ are occupied by solution characters (i.e., from Σ_1). For each distinct i and i', the instances of $\mathscr{C}$ from S_{i1} and $S_{i'1}$ match $\mathscr{C}$ at distinct sets of positions. Consider the k instances of $\mathscr{C}$ from S_{i1}, $1 \leqslant i \leqslant k$. The sets of positions in $\mathscr{C}$ that are matched by these instances must collectively match all positions of $\mathscr{C}$. Hence the corresponding subsets from $\mathscr{L}$ cover all elements of $\mathscr{B}$. □

Theorem 20. CAS$(m, |\Sigma|)$ *is* $W[2]$*-hard.*

Proof. Follows from Lemma 19 and the $W[2]$-hardness of SET COVER [3]. □

3.4. Other hardness results

Theorem 21. CAS($|\Sigma|$) *is not in* XP *unless* P = NP.

Proof. Follows from the *NP*-hardness of CLOSEST STRING (and hence CAS) when $|\Sigma| = 2$ [7] and Lemma 8. □

4. Membership results for 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* (layers of gates with unlimited fanin) are needed. 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 instance + modifications testing circuit*.

4.1. Center testing circuit

Let $\mathscr{F} = \{S_1, \ldots, S_m\}$ be an instance of CAS, and $\mathscr{C}$ is any center for $\mathscr{F}$. The jth length-l substring of S_i is denoted $S_i[j]$. The set X is be used to index size $l - d$ *subsequences* of a length-l string. Note that each X_p is an *ordered* $l - d$ tuple:

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

Let $A = \{a[i,j,p,q] : 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n - l + 1,\ 1 \leqslant p \leqslant |X|,\ 1 \leqslant q \leqslant l - d\}$ denote position q in $X_p \cap S_i[j]$. Let $B = \{b[u,v] : 1 \leqslant u \leqslant l, 1 \leqslant v \leqslant |\Sigma|\}$ be a set of boolean variables. The intended interpretation of variable $b[u,v]$ is that character v occupies position u in $\mathscr{C}$. The variable $a[i,j,p,q]$ 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 0.

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

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

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

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 $X_p = (1,2,3)$, 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 and if $X_p = (1,3,5)$, 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.

We claim that E has a weight l truth assignment if, and only if, there exists a center $\mathscr{C}$ for $\mathscr{F}$. If $\mathscr{C}$ exists, it is easy to verify that a truth assignment corresponding to $\mathscr{C}$ satisfies E. Conversely, let $\mathscr{T}$ be a weight l satisfying truth assignment for E. The clauses of E_1 ensure that $\mathscr{T}$ 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 $\mathscr{F}$.

Theorem 22. CAS$(l) \in W[2]$ *and* 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. □

4.2. 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 CAS 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 $B = \{b[i,j] : 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n-l+1\}$ be a set of boolean variables with the intended interpretation that $b[i,j]$ is set true when $S_i[j]$ is an instance of $\mathscr{C}$. Let $W = \{w[i,r,p] : 1 \leqslant i \leqslant m,\ 1 \leqslant r \leqslant d,\ 1 \leqslant p \leqslant l\}$ be a set of boolean variables with the intended meaning that any instance of $\mathscr{C}$ in S_i need not match $\mathscr{C}$ at position p. The index r is used to restrict the number of such *exemptions* to be not more than d for any instance. In the description of the circuit, the set of variables $A = \{a[i,j,p,q] : 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n-l+1,\ 1 \leqslant p \leqslant l,\ 1 \leqslant q \leqslant |\Sigma|\}$ act as an alias for the variables from B. As for the *center testing circuit*, 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 the value $false$.

Let $E = E_1E_2E_3$ be the boolean expression over the set of variables of $B \cup W$, where:

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

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

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

We claim that E has a weight $m + md$ satisfying truth assignment if, and only if, there is a center $\mathscr{C}$ for $\mathscr{F}$. Given center $\mathscr{C}$, a satisfying truth assignment for E can be obtained by setting $b[i,j]$ to true for each instance $S_i[j]$ of $\mathscr{C}$, and also setting $w[i,r,p]$ to true if the rth mismatch in the instance from S_i occurs at position p. For the converse case, let $\mathscr{T}$ be a weight $m + md$ satisfying truth assignment for E. The clauses of E_1 ensure that $\mathscr{T}$ 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 $\mathscr{T}$ to correspond directly to a set of instances and, for each instance, a set of positions where each instance may differ from a center. The fact that $\mathscr{T}$ satisfies E_3 implies that all instances agree in all positions with the possible and permitted exception of the exempted positions. Hence $\mathscr{F}$ has a center.

Theorem 23. $\textsc{cas}(m,d) \in W[3]$ *and* $\textsc{cas}(m,\Sigma,d) \in W[2]$.

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

4.3. 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 $\mathscr{F}$ (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 instance in each S_i, $2 \leqslant i \leqslant m$. The goal here is to have a weight $d+1$ truth assignment represent the selection of some j $(1 \leqslant j \leqslant n-l+1)$, 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:

$$X_1 = \{x_1[i,j,p,r] : 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n-l+1,\ 1 \leqslant p \leqslant l,\ 1 \leqslant r \leqslant |\Sigma|\},$$

$$X_2 = \{x_2[j] : 1 \leqslant j \leqslant n-l+1\},$$

$$X_3 = \{x_3[p,r] : 1 \leqslant p \leqslant l,\ 1 \leqslant r \leqslant |\Sigma|\}.$$

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 comes 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 $\mathscr{F}$. 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 $\{g[p,r]: 1 \leqslant p \leqslant l,\ 1 \leqslant r \leqslant |\Sigma|\}$ describes the "guessed" center, where

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

The lower layers of the circuit are described by the variables

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

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 $S_i[j]$:

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

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 1.

We now define the variables of B:

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

The circuit C is described by expression $E = E_1E_2E_3$ defined as:

$$E_1 = \prod_{1 \leqslant j < j' \leqslant n-l+1} (\neg x_2[j] + \neg x_2[j']),$$

$$E_2 = \prod_{p=1}^{l} \prod_{1 \leqslant r < r' \leqslant |\Sigma|} (\neg x_3[p,r] + \neg x_3[p,r']),$$

$$E_3 = \sum_{i=2}^{m} \sum_{j=1}^{n-l+1} \sum_{p=1}^{l} b[i,j,p].$$

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 $\mathscr{F}$. 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 24. CAS$(d) \in$ W[P].

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

5. Implications for common subsequence problems

Suitable parameterizations may also be used to extend results derived for one problem to related problems, and hence to highlight factors affecting the complexities of related problems. For example, consider the following extension of CAS stated relative to subsequences rather than substrings:

COMMON APPROXIMATE SUBSEQUENCE (CASEQ)

Instance: A set $\mathcal{F}=\{S_1,\ldots,S_m\}$ of strings over an alphabet Σ such that $|S_i| \leqslant n$, $1 \leqslant i \leqslant m$, and positive integers l and d such that $1 \leqslant l \leqslant n$ and $0 \leqslant d \leqslant l$.

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

Define g to be the maximum separation between any two symbols in a subsequence instance of center string $\mathscr{C}$ in any string in $\mathcal{F}$ (for example, given string $s=axxxbyyc$ and $\mathscr{C}=abc$, $g=3$). A number of common string problems can now be seen as subcases of, and hence are related to, CASEQ under appropriate restrictions of d and g, i.e.,

- CASEQ$(d=0,\ g=0) \Rightarrow$ LONGEST COMMON SUBSTRING
- CASEQ$(d \geqslant 0,\ g=0) \Rightarrow$ COMMON APPROXIMATE SUBSTRING
- CASEQ$(d=0,\ g \geqslant 0) \Rightarrow$ LONGEST COMMON SUBSEQUENCE (LCS)
- CASEQ$(d \geqslant 0, g \geqslant 0) \Rightarrow$ COMMON APPROXIMATE SUBSEQUENCE

All hardness results derived in this paper for CAS thus also hold for the corresponding parameterized versions of CASEQ; moreover, some of these variants are hard for higher levels of the W-hierarchy through hardness results for LCS [1,2]. It is an open problem whether any of the fixed-parameter tractability results for CAS or LCS apply to CASEQ. In any case, the above illustrates the relative importances of exact vs. approximate symbol occurrence and symbol position in common-string problems through the increase in complexity from LONGEST COMMON SUBSTRING, which is solvable in low-order polynomial time, to CASEQ, which is intractable in both polynomial-time and fixed-parameter settings.

The CASEQ problem can also be viewed as an extension of CAS, where the definition of "approximate" has been altered to allow limited insertions into the center string

$\mathscr{C}$. This definition can be further extended to allow for limited deletions from $\mathscr{C}$ as well.

6. Conclusions and future research

In this paper, we have derived the complexities of many parameterized variants of the COMMON APPROXIMATE SUBSTRING problem. These results, along with all results implied by the dependencies $d \leqslant l \leqslant n$, are summarized in Table 1. For fixed-parameter tractable variants, further research is necessary in order to find algorithms whose time and space requirements are most suited to the parameter restrictions inherent in specific applications. For example, optimal algorithms for probe and primer design, whose instances have larger lengths and smaller distances, are probably different from optimal algorithms for general motif finding, whose instances have smaller lengths but longer distances. Faster specialized algorithms can arise if other problem aspects are fixed in addition to the minimum set needed for fixed-parameter tractability [4,5,15].

Parameterized complexity analysis enables us to determine the effect of each aspect of a problem on that problem's complexity and to isolate those critical aspects or sets of aspects that can be fixed to yield useful algorithms for specific applications. In the case of CAS, fixing $|\Sigma|$ and l together produces fixed-parameter tractability, as does fixing n. Fixing m in addition to l does not produce a fixed-parameter tractable solution, but does reduce the complexity of the machinery needed to check solutions. While the proven results for the first two rows of Table 1 are different, they are not incompatible; fixing d, without restricting l, has as yet no proven impact on problem complexity. Several cells in the table have room for further investigation and potential improvement, such as the differences between hardness and membership classes for CAS(m,d) and CAS(d). Two variants, CAS$(|\Sigma|,d)$ and CAS$(m,|\Sigma|,d)$, have neither a hardness nor a fixed-parameter tractability result, making them prime candidates for future work. Their class membership is also a possible target for incremental improvement. We have also shown how the CAS hardness results can be transferred to a related problem that uses subsequences instead of substrings. This problem inherits many hardness results from both its substring and subsequence variants; a thorough investigation of it would extend this comprehensive parameterized analysis, and link the results here with previous results for finding common subsequences.

This investigation into the COMMON APPROXIMATE SUBSTRING problem has thus revealed the effects of various aspects and aspect interactions on that problem's complexity, as well as likely avenues for future development of exact algorithms.

References

[1] H. Bodlaender, R. Downey, M. Fellows, M. Hallett, H.T. Wareham, Parameterized complexity analysis in computational biology, Comput. Appl. Biosci. 11 (1) (1995) 49–57.

[2] H. Bodlaender, R. Downey, M. Fellows, H.T. Wareham, The parameterized complexity of sequence alignment and consensus, Theoret. Comput. Sci. 147 (1–2) (1995) 31–54.

[3] R. Downey, M. Fellows, Parameterized complexity, Monographs in Computer Science, Springer, New York, 1999.
[4] P.A. Evans, A.D. Smith, 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.
[5] P.A. Evans, H.T. Wareham, Practical algorithms for universal DNA primer design: an exercise in algorithm engineering (poster abstract), in: N. El-Mabrouk, T. Lengauer, D. Sankoff (Eds.), Currents in Computational Molecular Biology 2001, Montreal, PQ, 2001. Les Publications CRM, Montreal, pp. 25–26.
[6] M.R. Fellows, J. Gramm, R. Niedermeier, On the parameterized intractability of closest substring and related problems, in: H. Alt, A. Ferreira (Eds.), The 19th Internat. Symp. on Theoretical Aspects of Computer Science STACS 2002, Lecture Notes in Computer Science, Vol. 2285, Antibes/Juan-Les-Pins, France, 2002, Springer, Berlin, pp. 262–273.
[7] M. Frances, A. Litman, On covering problems of codes, Theory of Computing Systems 30 (2) (1997) 113–119.
[8] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of *NP*-Completeness, Freeman and Company, San Francisco, 1979.
[9] J. Gramm, R. Niedermeier, P. Rossmanith, Exact solutions for CLOSEST STRING and related problems, in: P. Eades, T. Takaoka (Eds.), Proc. 12th Ann. Symp. on Algorithms and Computation (ISAAC 2001), Lecture Notes in Computer Science, Vol. 2223, Springer, Berlin, 2001, pp. 441–453.
[10] T. Jiang, M. Li, On the approximation of shortest common supersequences and longest common subsequences, SIAM J. Comput. 24 (1995) 1122–1139.
[11] J.K. Lanctot, M. Li, B. Ma, S. Wang, L. Zhang, Distinguishing string selection problems, in: Proc. 10th Ann. ACM-SIAM Symp. on Discrete Algorithms, ACM Press, New York, 1999, pp. 633–642.
[12] M. Li, B. Ma, L. Wang, Finding similar regions in many sequences, in: ACM Symposium on Theory of Computing, ACM Press, New York, 1999, pp. 473–482.
[13] Bin Ma, A polynomial time approximation scheme for the closest substring problem, in: R. Giancarlo, D. Sankoff (Eds.), Combinatorial Pattern Matching (CPM 2000), Lecture Notes in Computer Science, Vol. 1848, Springer, Berlin, 2000, pp. 99–107.
[14] D. Maier, The complexity for some problems on subsequences and supersequences, J. Assoc. Comput. Mach. 25 (1978) 322–336.
[15] L. Marsan, M.-F. Sagot, Extracting structured motifs using a suffix tree—algorithms and application to promoter consensus identification, in: S. Minoru, R. Shamir (Eds.), Proc. Fourth Ann. Internat. Conf. Computational Molecular Biology (RECOMB'00), ACM Press, New York, 2000, pp. 210–219.
[16] P. Pevzner, S. Sze, Combinatorial approaches to finding subtle signals in DNA sequences, in: Bourne et al. (Ed.), Eighth Internat. Symp. on Intelligent Systems for Molecular Biology (ISMB'00), AAAI Press, Menlo Park, CA, 2000, pp. 269–278.
[17] H.T. Wareham, Systematic parameterized complexity analysis in computational phonology, Ph.D. Thesis, Department of Computer Science, University of Victoria, 1999.
[18] M.S. Waterman, R. Arratia, D.J. Galas, Pattern recognition in several sequences: consensus and alignment, Bull. Math. Biol. 46 (1984) 515–527.