COMBINATORICA
Bolyai Society – Springer-Verlag

COMBINATORICA **26** (2) (2006) 141–167

ON THE PARAMETERIZED INTRACTABILITY OF MOTIF SEARCH PROBLEMS*

MICHAEL R. FELLOWS, JENS GRAMM†, ROLF NIEDERMEIER‡

Received March 14, 2003

We show that *Closest Substring*, one of the most important problems in the field of consensus string analysis, is W[1]-hard when parameterized by the number k of input strings (and remains so, even over a binary alphabet). This is done by giving a "strongly structure-preserving" reduction from the graph problem *Clique* to *Closest Substring*. This problem is therefore unlikely to be solvable in time $O(f(k)\cdot n^c)$ for any function f of k and constant c independent of k, i.e., the combinatorial explosion seemingly inherent to this NP-hard problem cannot be restricted to parameter k. The problem can therefore be expected to be intractable, in any practical sense, for $k \geq 3$. Our result supports the intuition that *Closest Substring* is computationally much harder than the special case of *Closest String*, although both problems are NP-complete. We also prove W[1]-hardness for other parameterizations in the case of unbounded alphabet size. Our W[1]-hardness result for *Closest Substring* generalizes to *Consensus Patterns*, a problem arising in computational biology.

Mathematics Subject Classification (2000): 03D15, 68Q17, 68Q25

* An extended abstract of this paper was presented at the 19th International Symposium on Theoretical Aspects of Computer Science (STACS 2002), Springer-Verlag, LNCS 2285, pages 262–273, held in Juan-Les-Pins, France, March 14–16, 2002.

† Work was supported by the Deutsche Forschungsgemeinschaft (DFG), research project "OPAL" (optimal solutions for hard problems in computational biology), NI 369/2.

‡ Work was done while the author was with Wilhelm-Schickard-Institut für Informatik, Universität Tübingen. Work was partially supported by the Deutsche Forschungsgemeinschaft (DFG), Emmy Noether research group "PIAF" (fixed-parameter algorithms), NI 369/4.

1. Introduction

Searching common motifs is a central problem of consensus analysis based on strings (with, in particular, applications in computational biology [5,23, 25,26,32,33]). Two core problems in this context are *Closest Substring* [26] and *Consensus Patterns* [25]:

Input. k strings $s_1, s_2, \ldots, s_k$ over alphabet Σ and non-negative integers d and L.

Question in case of *Closest Substring*. Is there a string s of length L and, for all $i=1,\ldots,k$, a length-L substring s'_i of s_i such that $d_H(s,s'_i)\leq d$? (Here $d_H(s,s'_i)$ denotes the Hamming distance between s and s'_i.)

Question in case of *Consensus Patterns*. Is there a string s of length L and, for all $i=1,\ldots,k$, a length-L substring s'_i of s_i such that $\sum_{i=1}^{k} d_H(s,s'_i)\leq d$?

What is currently known about these two problems is summarized as follows.

The *Closest Substring* Problem

1. *Closest Substring* is NP-complete, and remains so for the special case of the *Closest String* problem, where the string s that we search for is of same length as the input strings. *Closest String* is NP-complete even for the further restriction to a binary alphabet [18,23].
2. On the positive side, both *Closest Substring* and *Closest String* admit polynomial time approximation schemes (PTAS's), where the objective function is the minimum Hamming distance d [25,26].
3. In the PTAS's for both *Closest String* and *Closest Substring*, the exponent of the polynomial bounding the running time depends on the goodness of the approximation. These are not efficient PTAS's (EPTAS's) in the sense of Cesati and Trevisan [6] and therefore are probably not useful in practice. Whether EPTAS's are possible for these approximation problems, or whether they are W[1]-hard also with respect to the distance parameter currently remains open.
4. *Closest String* is *fixed-parameter tractable* with respect to the parameter d, and can be solved in time $O(kL+kd\cdot d^d)$ [21].
5. *Closest String* is also fixed-parameter tractable with respect to the parameter k [21], but here the exponential parametric function is much faster growing, and the algorithm is probably of less practical use (see, however, [20] for some encouraging experimental results also in this case).

The *Consensus Patterns* Problem

1. *Consensus Patterns* is NP-complete and remains so for the restriction to a binary alphabet [25].
2. *Consensus Patterns* admits a PTAS [25], where the objective function is the minimum Hamming distance d.
3. The known PTAS for *Consensus Patterns* is not an EPTAS, and whether EPTAS's are possible, or whether PTAS approximation for this objective function is W[1]-hard, is an important issue that also currently remains open.

The key distinguishing point between *Closest Substring* and *Consensus Patterns* lies in the definition of the distance measure d between the "solution" string s and the substrings of the k input strings. Whereas *Closest Substring* uses a maximum distance metric, *Consensus Patterns* uses the sum of distances metric. This is of particular importance when discussing values of parameter d occurring in practice. Whereas it makes good sense for many applications to assume that d is a fairly small number in case of *Closest Substring*, this is much less reasonable in the case of *Consensus Patterns*. This will be of some importance when discussing our result for *Consensus Patterns*.

Many algorithms applied in practice try to solve motif search problems exactly, often using enumerative approaches in combination with heuristics [2,5,33]. In this paper, we explore the parameterized complexity of the basic motif problems in the framework of [12].

Concerning exact (parameterized) algorithms, we only briefly mention that, e.g., Sagot [35] studies motif discovery by solving *Closest Substring*, Evans and Wareham [13] give FPT algorithms for the same problem, and Blanchette *et al.* [2] developed a so-called phylogenetic footprinting method for a slightly more general version of *Consensus Patterns*. All these results, however, make essential use of the parameter "substring length" L and the running times show exponential behavior with respect to L. Hence, independently, Evans *et al.* [14] also developed several W[1]-hardness result. By way of contrast, these results heavily rely on the less interesting case of unbounded alphabet size, whereas our main results even hold for binary alphabet. Moreover, Evans *et al.* only deal with *Closest Substring* (there called *Common Approximate Substring*), whereas we extend our considerations and results to *Consensus Patterns*. To circumvent the computational limitations for larger values of L, many heuristics were proposed, e.g., Pevzner and Sze [33] present algorithms called WINNOWER (with respect to *Closest Substring*) and SP-STAR (with respect to *Consensus Patterns*), and Buhler

and Tompa [5] use random projections to find closest substrings. Our analysis makes a first step towards showing that, for exact solutions, we have to include L in the exponential growth; namely, we show that it is highly unlikely to find algorithms with a running time exponential *only* in k.

Our Main Results

Our main results are negative ones: we show that *Closest Substring* and *Consensus Patterns* are W[1]-hard with respect to the parameter k of the number of input strings, even in case of a binary alphabet. The main contribution is the development of a sophisticated, "parameter-preserving" reduction of *Clique* to *Closest Substring*.

For unbounded alphabet size, we show that the problems are W[1]-hard for the combined parameters L, d, and k. In the case of constant alphabet size, the complexity of the problems remain open when parameterized by d and k together, or by d alone. Note that in the case of *Consensus Patterns* our result gains particular importance, because here the distance parameter d usually is not small, whereas assuming that k is small is reasonable. Until now, it was known only that if one additionally considers the substring length L as a parameter, then running times exponential in L can be achieved [2,13,35]. An overview on known parameterized complexity results for *Closest Substring* and *Consensus Patterns* is given in Table 1.[1]

parameter	constant size alphabet	unbounded alphabet
d	?	W[1]-hard$^{(*)}$
k	W[1]-hard$^{(*)}$	W[1]-hard$^{(*)}$
d, k	?	W[1]-hard$^{(*)}$
L	FPT	W[1]-hard$^{(*)}$
d, k, L	FPT	W[1]-hard$^{(*)}$

Table 1. Overview on the parameterized complexity of *Closest Substring* and *Consensus Patterns* with respect to different parameterizations, where k is the number of given strings, L is the length of the substrings we search for, and d is the Hamming distance allowed. Results from this paper are marked by $(*)$. The FPT results for constant size alphabet can be achieved by enumerating all length L strings over Σ. Open questions are indicated by a question mark.

We achieve our results by giving parameterized many-one reductions from the W[1]-complete graph problem *Clique* to the respective problems. It is

[1] Note that for unbounded alphabet size similar results were obtained by Evans *et al.* [14].

important here to note that parameterized reductions are much more fine-grained and, from a combinatorial point of view, more structure-preserving than conventional polynomial-time reductions used in NP-completeness proofs, since parameterized reductions have to take care of the parameters. Establishing that *Closest Substring* and *Consensus Patterns* are W[1]-hard with respect to the parameter k requires significantly more technical effort than the already known demonstrations of NP-completeness. Finally, our work gives strong theory-based support for the common intuition that *Closest Substring* (W[1]-hard) seems to be a much harder problem than *Closest String* (in FPT [21]). Notably, this could *not* be expressed by "classical complexity measures," since both problems are NP-complete as well as both do have a PTAS.

Recently, based on the constructions presented in this paper, the slightly more general *Distinguishing Substring Selection* problem [9,23] was shown to be W[1]-hard also with respect to the distance parameters [19]. In particular, this implies that the recently presented PTAS for *Distinguishing Substring Selection* [9] cannot be improved into an EPTAS unless FPT = W[1] (see [19] and [6,10,15] for details). The corresponding question remains open for *Closest Substring* and *Consensus Patterns*.

Our work is organized as follows. In Section 2, we provide some background on parameterized complexity theory and we give a brief overview on related computational biology results. Afterwards, in Section 3, we present a parameterized reduction of *Clique* to *Closest Substring* in case of unbounded input alphabet size. Then, in Section 4, this is specialized to the case of binary input alphabet. Finally, Section 5 gives similar constructions and results for *Consensus Patterns* and the paper concludes with a brief summary and open questions in Section 6.

2. Preliminaries

In this section, we start with a brief introduction to parameterized complexity (more details can be found in the monographs [12,30] and the survey articles [1,10,15–17,29]).

Given an undirected graph $G=(V,E)$ with vertex set V, edge set E, and a positive integer k, the NP-complete *Vertex Cover* problem is to determine whether there is a subset of vertices $C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in C. *Vertex Cover* is *fixed-parameter tractable*. There now are algorithms solving it in time less than $O(kn + 1.3^k)$ [7,31]. The corresponding complexity class of parameterized problems solvable in deterministic time $f(k)\cdot n^{O(1)}$ – where f is an arbitrary

computable function only depending on parameter k and n is the problem size – is called FPT. By way of contrast, consider the NP-complete *Clique* problem: Given an undirected graph $G = (V, E)$ and a positive integer k, *Clique* asks whether there is a subset of vertices $C \subseteq V$ with at least k vertices such that C forms a clique by having all possible edges between the vertices in C. *Clique* appears to be *fixed-parameter intractable*: It is *not* known whether it can be solved in time $f(k) \cdot n^{O(1)}$.

The best known algorithm solving *Clique* runs in time $O(n^{ck/3})$ [28], where c is the exponent in the time bound for multiplying two integer $n \times n$ matrices (currently best known, $c = 2.38$, see [8]). The decisive point is that k appears in the exponent of n, and there seems to be no way "to shift the combinatorial explosion only into k," independent from n.

Downey and Fellows developed a completeness program for showing parameterized intractability [12]. However, the completeness theory of parameterized intractability involves significantly more technical effort (as will also become clear when following the proofs presented in this paper). We very briefly sketch some integral parts of this theory in the following.

Let $L, L' \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized languages.[2] For example, in the case of *Clique*, the first component is the input graph coded over some alphabet Σ and the second component is the positive integer k, that is, the parameter. We say that L *reduces to* L' *by a standard parameterized* m*-reduction* if there are functions $k \mapsto k'$ and $k \mapsto k''$ from $\mathbb{N}$ to $\mathbb{N}$ and a function $(x, k) \mapsto x'$ from $\Sigma^* \times \mathbb{N}$ to Σ^* such that

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

Notably, most reductions from classical complexity turn out *not* to be parameterized ones. The basic reference degree for parameterized intractability, W[1], can be defined as the class of parameterized languages that are equivalent to the *Short Turing Machine Acceptance* problem (also known as the *k-Step Halting* problem). Here, we want to determine, for an input consisting of a nondeterministic Turing machine M (with unbounded nondeterminism and alphabet size), and a string x, whether M has a computation path accepting x in at most k steps. This can trivially be solved in time $O(n^{k+1})$ by exploring all k-step computation paths exhaustively, and we would be surprised if this can be much improved.

Therefore, this is the parameterized analogue of the *Turing Machine Acceptance* problem that is the basic generic NP-complete problem in classical

[2] In general, the second component (representing the parameter) can also be drawn from Σ^*; for most cases, and, in particular, in this paper, assuming the parameter to be a positive integer is sufficient.

complexity theory, and the conjecture that $\mathrm{FPT} \neq \mathrm{W}[1]$ is very much analogous to the conjecture that $\mathrm{P} \neq \mathrm{NP}$. Other problems that are W[1]-*complete* (there are many) include *Clique* and *Independent Set*, where the parameter is the size of the relevant vertex set [11,12].

From a practical point of view, W[1]-hardness gives a concrete indication that a parameterized problem with parameter k problem is unlikely to allow for an algorithm with a running time of the form $f(k) \cdot n^{O(1)}$.

There is a straightforward factor-2-approximation algorithm for *Closest Substring*, sketched as follows. We test, for each of the length-L substrings s'_1 of input string s_1, whether each of the strings s_i, $i = 2,3,\ldots,k$, has a length-L substring s'_i with $d_H(s'_1, s'_i) \leq 2d$. Note that for an optimal solution string s which corresponds to matching substrings s'_i in s_i, $i = 2,3,\ldots,k$, s'_1 necessarily satisfies the property tested above. Therefore, if none such s'_1 exists, the given instance has no solution. If we find at least one such s'_1 then we output solution string $s := s'_1$. Since s'_1 satisfies $d_H(s'_1, s'_i) \leq 2d$ for $i = 1,2,\ldots,k$, this algorithm is a factor-2-approximation.

The first better-than-2 approximation with factor $2 - 2/(2|\Sigma| + 1)$ was given by Li *et al.* [24]. Finally, there are PTAS's for *Consensus Patterns* [25] as well as for *Closest Substring* [26], both of which, however, have impractical running times.

3. *Closest Substring*: Unbounded Alphabet

We first describe a reduction from the W[1]-hard *Clique* problem to *Closest Substring* which is a parameterized m-reduction with respect to the aggregate parameter (L, d, k) in case of unbounded alphabet size.

3.1. Reduction of *Clique* to *Closest Substring*

A *Clique* instance is given by an undirected graph $G = (V, E)$, with a set $V = \{v_1, v_2, \ldots, v_n\}$ of n vertices, a set E of m edges, and a positive integer k denoting the desired clique size. We describe how to generate a set S of $\binom{k}{2}$ strings such that G has a clique of size k iff there is a string s of length $L := k+1$ such that every $s_i \in S$ has a substring s'_i of length L with $d_H(s, s'_i) \leq d := k-2$. If a string $s_i \in S$ has a substring s'_i of length L with $d_H(s, s'_i) \leq d$, we call s'_i a *match* for s. We assume $k > 2$, because $k = 1,2$ are trivial cases.

Alphabet. The alphabet of the produced instance is given by the disjoint union of the following sets:

- $\{[v_i] \mid v_i \in V\}$, i.e., an alphabet symbol for every vertex of the input graph; we call them *encoding symbols*;
- $\{[c_{i,j}] \mid i = 1,\ldots,k, j = i+1,\ldots,k\}$, i.e., a unique symbol for every of the $\binom{k}{2}$ produced strings; we call them *string identification symbols*;
- $\{\#\}$ which we call the *synchronizing symbol*.

This makes a total of $n + \binom{k}{2} + 1$ alphabet symbols.

Choice strings. We generate a set of $\binom{k}{2}$ *choice strings* $S_c = \{c_{1,2},\ldots,c_{1,k}, c_{2,3}, c_{2,4},\ldots,c_{k-1,k}\}$ and we assume that the strings in S_c are ordered as shown. *Every* choice string will encode the whole graph; it consists of m concatenated strings, each of length $k+1$, called *blocks*; by this, we have one block for every edge of the graph. The blocks will be separated by *barriers*, which are length k strings consisting of k identification symbols corresponding to the respective string. A choice string $c_{i,j}$ is given by

$$c_{i,j} := \langle \text{block}(i,j,e_1)\rangle\, ([c_{i,j}])^k\, \langle \text{block}(i,j,e_2)\rangle\, ([c_{i,j}])^k \ldots \ldots ([c_{i,j}])^k\, \langle \text{block}(i,j,e_m)\rangle,$$

where $e_1, e_2, \ldots, e_m$ are the edges of G and $\langle \text{block}()\rangle$ will be defined below. The solution string s will have length $k+1$, which is exactly the length of one block.

Block in a choice string. Every block is a string of length $k+1$ and it encodes an edge of the input graph. Every choice string contains a block for every edge of the input graph; different choice strings, however, encode the edges in different positions of their blocks: For a block in choice string $c_{i,j}$, positions i and j are called *active* and these positions encode the edge. Let e be the edge to be encoded and let e connect vertices v_r and v_s, $1 \le r < s \le n$. Then, the ith position of the block is $[v_r]$ in order to encode v_r and the jth position is $[v_s]$ in order to encode v_s. The last position of a block is set to the synchronizing symbol #. All remaining positions in the block are set to $c_{i,j}$'s identification symbol $[c_{i,j}]$. Thus, the block is given by

$$\langle \text{block}(i,j,(v_r,v_s))\rangle \;:=\; ([c_{i,j}])^{i-1}\, [v_r]\, ([c_{i,j}])^{j-i-1}\, [v_s]\, ([c_{i,j}])^{k-j}\, \#.$$

Values for L and d. We set $L := k+1$ and $d := k-2$.

Example 1. Let $G = (V,E)$ be an undirected graph with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{(v_1,v_3), (v_1,v_4), (v_2,v_3), (v_3,v_4)\}$ (as shown in Fig. 1(a)) and let $k = 3$. Using G, we exhibit the above construction of $\binom{k}{2} = 3$ choice strings $c_{1,2}$, $c_{1,3}$, and $c_{2,3}$ (as shown in Fig. 1(b)). We claim that (which will be proven in the following subsection) there exists a clique of size k in G iff there is a

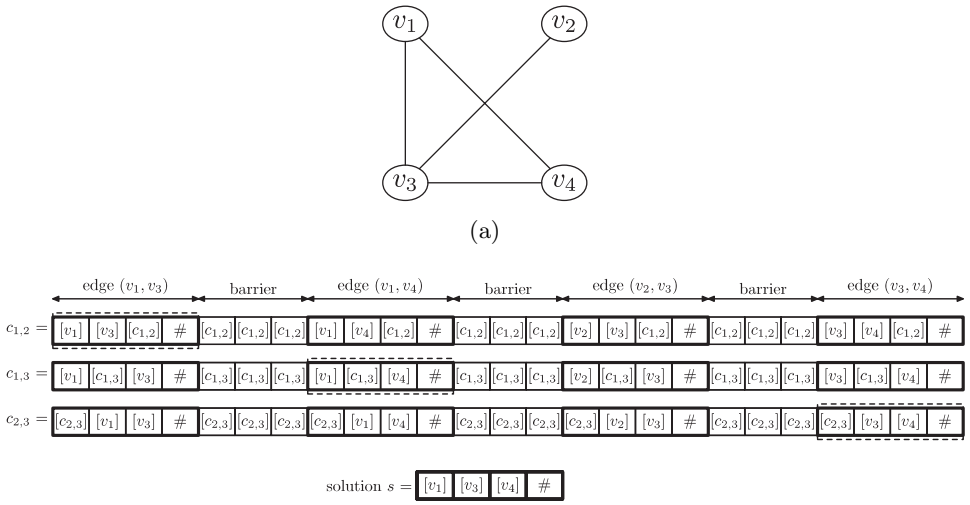

Fig. 1. Example for the reduction from a *Clique* instance G with $k=3$ (shown in (a)) to a *Closest Substring* instance with bounded alphabet (shown in (b)) as explained in Example 1. In (b), we display the constructed strings $c_{1,2}$, $c_{1,3}$, and $c_{2,3}$ (the contained blocks are highlighted by bold boxes) and the solution string s that is found, since G has a clique of size $k=3$; s is a string of length $k+1=4$ such that $c_{1,2}$, $c_{1,3}$, and $c_{2,3}$ have length 4 substrings (indicated by dashed boxes) that have Hamming distance at most $k-2=1$ to s.

string s of length $L := \binom{k}{2}+1 = 4$ such that, for $1 \le i < j \le 3$, each $c_{i,j}$ contains a length 4 substring $s_{i,j}$ with $d_H(s, s_{i,j}) \le d := k-2 = 1$.

The choice strings are over an alphabet consisting of $\{[v_1], [v_2], [v_3], [v_4]\}$ (the encoding symbols, i.e., one symbol for every vertex of G), $\{[c_{1,2}], [c_{1,3}], [c_{2,3}]\}$ (the string identification symbols), and $\{\#\}$ (the synchronizing symbol). Every string $c_{i,j}$, $1 \le i < j \le 3$, consists of four blocks, each of which encodes an edge of the graph. Every block is of length $\binom{k}{2}+1 = 4$ and has $\#$ at its last position. The blocks are separated by barriers consisting of $([c_{i,j}])^k = ([c_{i,j}])^3$.

In string $c_{1,2}$, positions 1 and 2 within a block are active and encode the corresponding edge (in general, in $c_{i,j}$ positions i and j within a block are active). All of the first k positions of a block in string $c_{i,j}$ which are not active contain the $[c_{i,j}]$ symbol. Thus, e.g., the block in $c_{1,2}$ encoding the edge (v_1, v_3) is given by $[v_1][v_3][c_{1,2}]\#$. Further details can be found in Fig. 1.

The closest substring that corresponds to the k-clique in G consisting of vertices v_1, v_3, and v_4 is $[v_1][v_3][v_4]\#$. The corresponding matches are

$[v_1][v_3][c_{1,2}]\#$ in $c_{1,2}$ (encoding the edge (v_1, v_3)), $[v_1][c_{1,3}][v_4]\#$ in $c_{1,3}$ (encoding the edge (v_1, v_4)), and $[c_{2,3}][v_3][v_4]\#$ in $c_{2,3}$ (encoding the edge (v_3, v_4)).

3.2. Correctness of the Reduction

To prove the correctness of the proposed reduction, we have to show an equivalence, consisting of two directions. The easier one is to see that a k-clique implies a closest substring fulfilling the given requirements.

Proposition 1. *For a graph with a k-clique, the construction in Subsection 3.1 produces an instance of* Closest Substring *which has a solution, i.e., there is a string s of length L such that every $c_{i,j} \in S_c$ has a substring $s_{i,j}$ with $d_H(s, s_{i,j}) \leq d$.*

Proof. Let the input graph have a clique of size k. Let $h_1, h_2, \ldots, h_k$ denote the indices of the clique's vertices, $1 \leq h_1 < h_2 < \ldots < h_k \leq n$. Then, we claim that a solution for the produced *Closest Substring* instance is

$$s := [v_{h_1}][v_{h_2}] \ldots [v_{h_k}]\#.$$

Consider choice string $c_{i,j}$, $1 \leq i < j \leq k$. As the vertices $v_{h_1}, v_{h_2}, \ldots, v_{h_k}$ form a clique, we have an edge connecting v_{h_i} and v_{h_j}. Choice string $c_{i,j}$ contains a block $s_{i,j} := \langle \text{block}(i, j, (v_{h_i}, v_{h_j})) \rangle$ encoding this edge:

$$s_{i,j} := ([c_{i,j}])^{i-1}\,[v_{h_i}]\,([c_{i,j}])^{j-i-1}\,[v_{h_j}]\,([c_{i,j}])^{k-j}\#,$$

We have $d_H(s, s_{i,j}) = k - 2$, and we can find such a block for every $c_{i,j}$, $1 \leq i < j \leq k$. ∎

For the reverse direction, we show in Proposition 2 that a solution in the produced *Closest Substring* instance implies a k-clique in the input graph. For this, we need the following two lemmas, which show that a solution to the instance constructed in Subsection 3.1 has encoding symbols at its first k positions and the synchronizing symbol # at its last position.

Lemma 1. *A closest substring s contains at least two encoding symbols and at least one synchronization symbol.*

Proof. Let s be a solution of the *Closest Substring* instance produced by the construction in Subsection 3.1. Let $A_{[c]}(s)$ be the set of string identification symbols from $\{\,[c_{i,j}] \mid 1 \leq i < j \leq k\,\}$ that occur in s. Let $S'_{[c]}(s) \subseteq S_c$ be the subset of choice strings that do *not* contain a symbol from $A_{[c]}(s)$.

Since s is of length $k+1$, we have $|A_{[c]}(s)| \leq k+1$. Therefore, for $k \geq 4$, there are at least $\binom{k}{2} - (k+1)$ choice strings in $S'_{[c]}(s)$. We show that with less than two encoding symbols and no synchronizing symbol, we cannot find matches for s (with maximally allowed Hamming distance $d = k-2$) in the choice strings of $S'_{[c]}(s)$. Observe that, in every choice string, because of the barriers, every length $k+1$ substring contains at most two encoding symbols and at most one symbol #. Observe further that, taken a choice string from $S'_{[c]}(s)$, positions with symbols from $\{[c_{i,j}] \mid 1 \leq i < j \leq k\}$ cannot coincide with the corresponding positions in s. Therefore, s has a match in such a string only if s has two encoding symbols and one symbol # that all coincide with the corresponding positions in the selected substring. This proves the claim for $k \geq 4$. Regarding $k = 3$, if $|A_{[c]}(s)| < 3$, then the above argument applies here, too. If, however, $|A_{[c]}(s)| = 3$, a length 4 substring in every choice string has at least two positions that do not coincide with the corresponding positions in s. ∎

Based on Lemma 1, we can now exactly specify the numbers and positions of the encoding and synchronizing symbols in the closest substring.

Lemma 2. *A closest substring s contains encoding symbols at its first k positions and a symbol # at its last position.*

Proof. Let $n_{\#}(s)$ denote the number of symbols # in s, let $n_{[c]}(s)$ denote the number of string identification symbols in s, and let $n_{[v]}(s)$ denote the number of encoding symbols in s. Let $S'_{[c]}(s) \subseteq S_c$ be the subset of choice strings whose string identification symbol does not occur in s. In the following, we establish a lower bound on the number of strings in $S'_{[c]}(s)$ and an upper bound on the number of strings from $S'_{[c]}(s)$ in which we can find a match for s. Comparing these bounds, we will show that, if $n_{\#}(s) > 1$, then there are choice strings in $S'_{[c]}(s)$ in which we cannot find a match; we will conclude that $n_{\#}(s) = 1$. Then, we will show that, if $n_{[v]}(s) < k$, then again there are strings in $S'_{[c]}(s)$ without a match for s; we will conclude that $n_{[v]}(s) = k$.

Regarding the size of $S'_{[c]}(s)$, a lower bound on its size is $|S'_{[c]}(s)| \geq \binom{k}{2} - n_{[c]}(s)$. To obtain an upper bound on the number of strings from $S'_{[c]}(s)$ in which we can find a match for s, we recall that such matches must contain two encoding symbols and one symbol # that all coincide with the corresponding positions in s. On the one hand, the synchronizing symbol of a block must coincide with a symbol # in s. On the other hand, in all blocks of a choice string, its encoding symbols are in fixed positions relative

to the block's synchronizing symbol, e.g., in choice string $c_{1,2}$, the encoding symbols are located only at the first and second position and # at the last position of a block in $c_{1,2}$. For these two reasons, one symbol # in s can provide matches in at most $\binom{n_{[v]}(s)}{2}$ choice strings from $S'_{[c]}(s)$. Consequently, $n_{\#}(s)$ many symbols # in s can provide matches in at most $n_{\#}(s) \cdot \binom{n_{[v]}(s)}{2}$ choice strings from $S'_{[c]}(s)$.

Summarizing, we have at least $\binom{k}{2} - n_{[c]}(s)$ choice strings in $S'_{[c]}(s)$ and we can find matches in at most $n_{\#}(s) \cdot \binom{n_{[v]}(s)}{2}$ many of them. Thus, we find matches for s in all choice strings only if

$$n_{\#}(s) \cdot \binom{n_{[v]}(s)}{2} \geq \binom{k}{2} - n_{[c]}(s). \tag{1}$$

In order to show that s contains exactly one synchronizing symbol, we assume that $n_{\#}(s) > 1$ (we know that $n_{\#}(s) \geq 1$ by Lemma 1) while $k > 2$, and show that inequality (1) is violated.

We know that $k+1 = n_{[v]}(s) + n_{[c]}(s) + n_{\#}(s)$ and, by Lemma 1, that $n_{[v]}(s) \geq 2$. Using these, we conclude, on the one hand, that $n_{\#}(s) \cdot \binom{n_{[v]}(s)}{2} \leq n_{\#}(s) \cdot \binom{k+1-n_{\#}(s)}{2}$ and, since $n_{\#}(s) \geq 2$, that $n_{\#}(s) \cdot \binom{k+1-n_{\#}(s)}{2} \leq 2 \cdot \binom{k-1}{2}$. On the other hand, we have that $\binom{k}{2} - n_{[c]}(s) \geq \binom{k}{2} - (k-1-n_{\#}(s))$ and, since $n_{\#}(s) \geq 2$, $\binom{k}{2} - (k-1-n_{\#}(s)) \geq \binom{k}{2} - (k-3)$. For $k \geq 3$, however we have $\binom{k}{2} - (k-3) > 2 \cdot \binom{k-1}{2}$. Thus,

$$\begin{aligned} n_{\#}(s) \cdot \binom{n_{[v]}(s)}{2} \leq n_{\#}(s) \cdot \binom{k+1-n_{\#}(s)}{2} &< \binom{k}{2} - (k-1-n_{\#}(s)) \\ &\leq \binom{k}{2} - n_{[c]}(s), \end{aligned}$$

i.e., there are choice strings in $S'_{[c]}(s)$ which contain no match for s, a contradiction. Since (Lemma 1) $n_{\#}(s) \geq 1$, we conclude that $n_{\#}(s) = 1$.

In order to show that s contains exactly k encoding symbols, we assume that $n_{[v]}(s) < k$ while $k > 2$ and $n_{\#}(s) = 1$, and show that inequality (1) is violated. Since $k+1 = n_{[v]}(s) + n_{[c]}(s) + n_{\#}(s) = n_{[v]}(s) + n_{[c]}(s) + 1$, we have $\binom{k}{2} - n_{[c]}(s) = \binom{k}{2} - (k - n_{[v]}(s))$ and, thus,

$$\binom{n_{[v]}(s)}{2} < \binom{k}{2} - (k - n_{[v]}(s)) = \binom{k}{2} - n_{[c]}(s),$$

i.e., again, some strings in $S'_{[c]}(s)$ have no match for s, a contradiction. Thus, on the one hand, we have $n_{[v]}(s) \geq k$, and, on the other hand, we have $n_{\#}(s) = 1$ and, therefore, $n_{[v]}(s) \leq k$.

Note that, if an encoding symbol is located *after* the synchronizing symbol in s, then, due to the barriers, it is not possible that both # and this encoding symbol coincide with the respective positions in every choice string from $S'_{[c]}(s)$, e.g., in $c_{1,2}$. Therefore, symbol # is located at the last position of s. ∎

Proposition 2. *The first k characters of a closest substring correspond to k vertices of a clique in the input graph.*

Proof. By Lemma 2, a closest substring s has encoding symbols at its first k positions and a synchronizing symbol at its last position. Consequently, the blocks are the only possible matches of s in the choice string. Now, assume that $s = [v_{h_1}][v_{h_2}] \ldots [v_{h_k}]\#$ for $h_1, h_2, \ldots, h_k \in \{1, \ldots, n\}$. Consider any two h_i, h_j, $1 \leq i < j \leq k$, and choice string $c_{i,j}$. Recall that in this choice string, the blocks encode edges at their ith and jth position, they have # at their last position, and all their other positions are set to a string identification symbol unique for this choice string. Thus, we can only find a block that is a match if there is a block with $[v_{h_i}]$ at its ith position and $[v_{h_j}]$ at its jth position. We have such a block only if there is an edge connecting v_{h_i} and v_{h_j}. Summarizing, the closest substring s implies that there is an edge between every pair of $\{v_{h_1}, v_{h_2}, \ldots, v_{h_k}\}$; these vertices form a k-clique in the input graph. ∎

Propositions 1 and 2 establish the following hardness result. Note that hardness for the combination of all three parameters also implies hardness for each subset of the three.

Theorem 1. Closest Substring *with unbounded alphabet is* W[1]*-hard for every combination of the parameters L, d, and k.*

4. *Closest Substring*: Binary Alphabet

We modify the reduction from Section 3 to achieve a *Closest Substring* instance with binary alphabet proving a W[1]-hardness result also in this case. In contrast to the previous construction, we cannot encode every vertex with its own symbol and we cannot use a unique symbol for every produced string. Also, we have to find new ways to "synchronize" the matches of our solution, a task previously done by the synchronizing symbol "#". To overcome these problems, we construct an additional "complement string" for

the input instance and we lengthen the blocks in the produced choice strings considerably.

4.1. Reduction of *Clique* to *Closest Substring*

Number strings. To encode integers between 1 and n, we introduce *number strings* $\langle \text{number}(pos) \rangle$, which have length n and which have symbol "1" at position pos and symbol "0" elsewhere: $0^{pos-1}10^{n-pos}$. In contrast to the reduction from Section 3, now we use these number strings to encode the vertices of a graph.

Choice strings. As in Section 3, we generate a set of $\binom{k}{2}$ *choice strings* $S_c = \{c_{1,2}, c_{1,3}, \ldots, c_{k-1,k}\}$. Again, every choice string will consist of m *blocks*, one block for every edge of the graph. The choice string $c_{i,j}$ is given by

$$c_{i,j} \;:=\; \langle \text{block}(i,j,e_1) \rangle \langle \text{block}(i,j,e_2) \rangle \ldots \langle \text{block}(i,j,e_m) \rangle,$$

where $e_1, e_2, \ldots, e_m$ are the edges of the input graph and $\langle \text{block}() \rangle$ is defined below. The length of a closest substring will be exactly the length of one block.

Block in a choice string. Every block consists of a front tag, an encoding part, and a back tag. A block in choice string $c_{i,j}$ encodes an edge e; let e be an edge connecting vertices v_r and v_s, $1 \le r < s \le n$, and let $c_{i,j}$ be the (according to the given order) i'th string in S_c. Then, the corresponding block is given by

$$\langle \text{block}(i,j,(v_r,v_s)) \rangle \;:=\; \langle \text{front\_tag} \rangle \langle \text{encode}(i,j,(v_r,v_s)) \rangle \langle \text{back\_tag}(i') \rangle.$$

Front tags. We want to enforce that a closest substring can only match substrings at certain positions in the produced choice strings, using front tags:

$$\langle \text{front\_tag} \rangle \;:=\; (1^{3nk}0)^{nk},$$

i.e., a front tag has length $(3nk+1) \cdot nk$. By this arrangement, the closest substring s and every match of s start (as will be shown in Subsection 4.2) with the front tag.

Encoding part. The encoding part consists of k sections, each of length n. The encoding part corresponds to the blocks used in Section 3. As a consequence, in $\langle \text{block}(i,j,e) \rangle$ the ith and jth section are called *active* and encode

edge $e=(v_r,v_s)$, $1\le r<s\le n$; section i encodes v_r by $\langle \text{number}(r)\rangle$ and section j encodes v_s by $\langle \text{number}(s)\rangle$. The other sections except for i and j are called *inactive* and are given by $\langle \text{inactive}\rangle := 0^n$. Thus,

$$\langle \text{encode}(i,j,(v_r,v_s))\rangle := \\ (\langle \text{inactive}\rangle)^{i-1}\langle \text{number}(r)\rangle(\langle \text{inactive}\rangle)^{j-i-1}\langle \text{number}(s)\rangle(\langle \text{inactive}\rangle)^{k-j}.$$

Back tag. The back tag of a block is intended to balance the Hamming distance of the closest substring to a block, as will be explained later. The back tag consists of $\binom{k}{2}$ sections, each section has length $nk-2k+2$. The i'th section consists of symbols "1," all other sections consist of symbols "0":

$$\langle \text{back\_tag}(i')\rangle \;:=\; 0^{(i'-1)(nk-2k+2)}1^{nk-2k+2}0^{(\binom{k}{2}-i')(nk-2k+2)}$$

Template string. The set of choice strings is complemented by one *template string*. It consists, in analogy to the blocks in the choice strings, of three parts: A front tag of length $(3nk+1)\cdot nk$, followed by a length nk string of symbols "1," followed by a length $\binom{k}{2}(nk-2k+2)$ string of symbols "0." Thus, the template string has the same length as a block in a choice string, i.e., $(3nk+1)\cdot nk+nk+\binom{k}{2}(nk-2k+2)$.

Values for d and L. We set $L := (3nk+1)\cdot nk+nk+\binom{k}{2}(nk-2k+2)$ and $d:=nk-k$. As we will show in Subsection 4.2, the possible matches for a string of this length are the blocks in the choice strings, and, concerning the template string, the template string itself.

Notation. For a closest substring s, we denote its first $(3nk+1)\cdot nk$ symbols (the front tag) by s', the following nk symbols (its encoding part) by s'', and the last $\binom{k}{2}(nk-2k+2)$ symbols (its back tag), by s'''. Analogously, the three parts of the template string t are denoted t',t'', and t'''. A particular block of a choice string $c_{i,j}$, is referred to by $s_{i,j}$; its three parts are called $s'_{i,j},s''_{i,j}$, and $s'''_{i,j}$.

Example 2. Let $G=(V,E)$ be the graph from Example 1, with $V=\{v_1,v_2,v_3,v_4\}$ and $E=\{(v_1,v_3),(v_1,v_4),(v_2,v_3),(v_3,v_4)\}$ (as shown in Fig. 1(a)) and let $k=3$. In the following, we outline the above construction of $\binom{k}{2}=3$ choice strings c_1, c_2, and c_3 and one template string t over alphabet $\Sigma=\{0,1\}$ as displayed in Fig. 2.

Every string c_1, c_2, and c_3 consists of four blocks corresponding to the four edges of G. Fig. 2(a) displays the first block of c_1 corresponding to edge (v_1,v_3). It consists of a front tag, an encoding part, and a back tag. The front tag (not displayed in detail in the figure) is given by $\langle \text{front\_tag}\rangle := (1^{3nk}0)^{nk}=(1^{36}0)^{12}$; all front tags for all blocks in all constructed strings

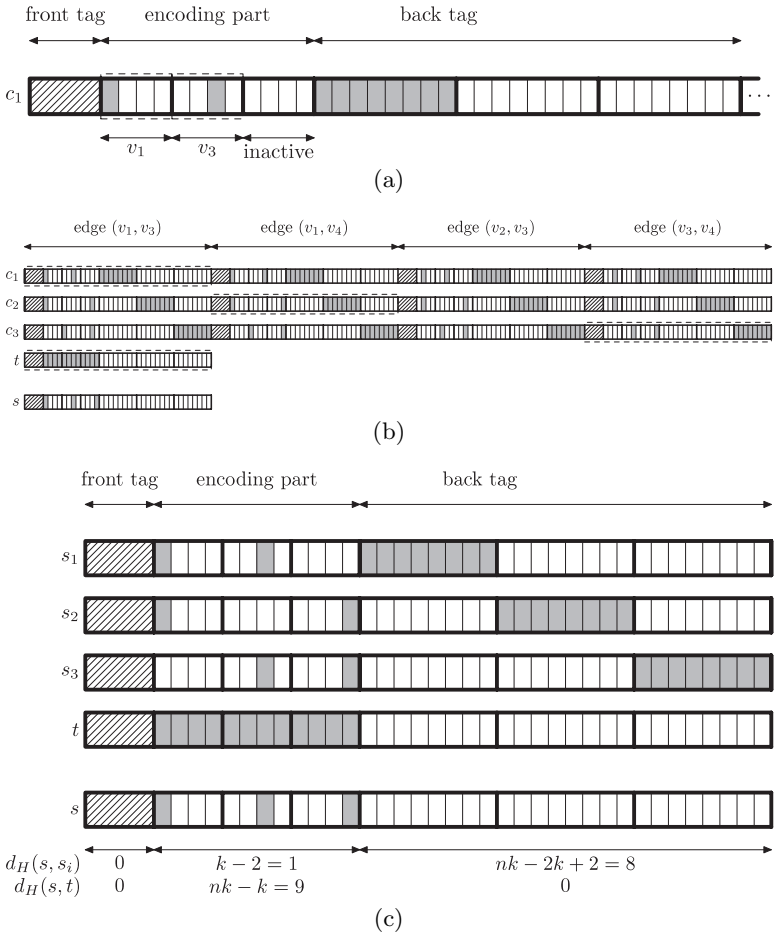

Fig. 2. Example for the reduction from the *Clique* instance G (shown in Fig. 1(a)) to a *Closest Substring* instance with binary alphabet as explained in Example 2. When displaying the strings, we omit the details of the front tag parts and only indicate them shortened in their proportion to the other parts of the strings; all front tag parts in all strings are equal. In the encoding parts and the back tag parts, we indicate the symbols "1" of the construction by dark boxes, the symbols "0" by white boxes. In (a), we outline the first block of c_1. In its encoding part, sections 1 and 2 (sections are indicated by bold separating lines) are active (indicated by dashed boxes) and encode the first edge (v_1, v_3) of graph G; the remaining third section is inactive. In its back tag part, the first section is filled with symbols "1." In (b), we give an overview on all constructed strings, the choice strings c_1, c_2, and c_3, and the template string t. We also display the closest substring s that is found, since G has a clique of size $k=3$; its matches in c_1, c_2, c_3, and t are indicated by dashed boxes. In (c), we focus on these matches and the solution string s. We state, separately for the front tag, the encoding, and the back tag part, the Hamming distances of s to a match s_i, $i=1,2,3$ (the distances are equal for s_1, s_2, and s_3) and to the template string t.

are the same. The back tag of the first block consists of $\binom{k}{2}$ sections; since the back tag is in the *first* string, the *first* section is filled with "1"s and the remaining sections are filled with "0"s. Thus, the back tag is given by $1^{nk-2k+2}0^{(\binom{k}{2}-1)(nk-2k+2)} = 1^8 0^{16}$, and all back tags for blocks in the first string are given like this. The encoding part consists of $k=3$ sections, each section of length $n=4$. In the blocks of string c_1, the first and the second section are active; in the first block they encode edge (v_1, v_3). Therefore, the first section is given by $\langle \text{number}(1) \rangle$ and the second one by $\langle \text{number}(3) \rangle$, the remaining inactive section is filled with "0"s.

Fig. 2(b) displays an overview on all constructed strings c_1, c_2, c_3, and t. In all strings, block i encodes the ith edge, $1 \le i \le 4$. However, the active sections of the encoding part and the back tags differ for different strings. The template string t consists only of one block, which has a front tag, a part corresponding to the encoding part, filled with "1"s, and a part corresponding to the back tag, filled with "0"s.

Since G has a k-clique for $k = 3$, consisting of vertices v_1, v_3, and v_4, we find a solution s for the constructed *Closest Substring* instance. This s has a front tag, and its back tag part is filled with "0" symbols. The encoding part encodes the vertices of the clique, it is given by $\langle \text{number}(1) \rangle \langle \text{number}(3) \rangle \langle \text{number}(4) \rangle$.

Fig. 2(c) gives a focus on the matches that are found in c_1, c_2, c_3, and t, which are, for the choice strings, referred to by s_1, s_2, and s_3, respectively. The front tag part s' has distance 0 to the front tags s'_1, s'_2, s'_3, and t'. The encoding part s'' contains $k=3$ many "1"s; s''_1, s''_2, s''_3 have two "1"s each and, in each case, these "1"s coincide with "1"s in s''. Therefore, $d_H(s'', s''_i) = k-2=1$, $1 \le i \le 3$. The encoding part of the template string, t'', only consists of "1"s and, therefore, $d_H(s'', t'') = nk-k=9$. The back tag s''' only consists of "0"s; each back tag s'''_1, s'''_2, and s'''_3 contains $nk-2k+2=8$ many "1"s; therefore $d_H(s''', s'''_i) = 8$, $1 \le i \le 3$. The back tag of the template string, t''', contains only "0"s and, hence, $d_H(s''', t''') = 0$. Altogether, this shows that, for $1 \le i \le 3$, $d_H(s, s_i) = d_H(s, t) = nk-k=9$ as required.

4.2. Correctness of the Reduction

To prove the correctness of the reduction, again the easier direction is to show that a k-clique implies a closest substring fulfilling the given requirements.

Proposition 3. *For a graph with a k-clique, the construction in Subsection 4.1 produces an instance of* Closest Substring *that has a solution, i.e.,*

there is a string s of length L such that every $c_{i,j} \in S_c$ has a length L substring $s_{i,j}$ with $d_H(s, s_{i,j}) \leq d$ and $d_H(s,t) \leq d$.

Proof. Let the graph have a clique of size k. Let $h_1, h_2, \ldots, h_k$ denote the indices of the clique's vertices, $1 \leq h_1 < h_2 < \ldots < h_k \leq n$. Then, we can find a closest substring s, consisting of three parts s', s'', and s''', as follows: its front tag s' is given by $\langle \text{front\_tag} \rangle$; its encoding part s'' is given by $\langle \text{number}(h_1) \rangle \langle \text{number}(h_2) \rangle \ldots \langle \text{number}(h_k) \rangle$; its back tag s''' is $0^{\binom{k}{2}(nk-2k+2)}$. It follows from the construction that the choice strings have substrings that are matches for this s: For every $1 \leq i < j \leq k$, we produced choice string $c_{i,j}$ with a block $s_{i,j}$ encoding the edge between vertices v_{h_i} and v_{h_j}. For these blocks as well as for the template string, the following table reports the distance they have to the solution string, separately for each of their three parts and in total:

$d_H(\cdot,\cdot)$	s'	s''	s'''	s
match $s_{i,j}$ in choice string $c_{i,j}$	0	$k-2$	$nk-2k+2$	$nk-k$
template string t	0	$nk-k$	0	$nk-k$

As is obvious from these distance values, the indicated substrings in the choice strings all have Hamming distance $d = nk-k$ to the solution string and, therefore, are matches for s. ∎

For the reverse direction, we assume that the *Closest Substring* instance has a solution. We need the following statements:

Lemma 3. *A solution s and all its matches in the input instance start with the front tag.*

Proof. Since s is of length $L = (3nk+1) \cdot nk + nk + \binom{k}{2}(nk-2k+2)$, the only possible match in the template string is the template string itself. Therefore, s' can differ from t' in at most $d = nk-k$ symbols. We can show that the only substrings in a choice string $c_{i,j}$ that are possible matches for s with Hamming distance at most d start with the front tag, as we argue in the following.

Since s is a solution, there is a match in $c_{i,j}$ and we denote it by $s_{i,j}$. Denote the the first $(3nk+1) \cdot nk$ symbols of $s_{i,j}$ by $s'_{i,j}$. Since $d_H(s', s'_{i,j}) \leq nk-k$ and $d_H(s', t') \leq nk-k$, we necessarily (triangle inequality for Hamming metric) have $d_H(s'_{i,j}, t') \leq 2(nk-k)$. We show that this is only possible when $s'_{i,j}$ coincides with a front tag of a block of $c_{i,j}$. Assuming that it does not, we will show that $d_H(s'_{i,j}, t') > 2(nk-k)$, a contradiction.

Firstly, assume that the starting position of $s'_{i,j}$ and the starting position of a front tag in $c_{i,j}$ differ by p positions, $1 \leq p \leq 3nk$. Then, at least $nk-1$

symbols "0" of t' are aligned with symbols "1" of the front tag in $s'_{i,j}$ and at least $nk-1$ symbols "1" of t' are aligned with symbols "0" of $s'_{i,j}$. This implies $d_H(s'_{i,j},t') > 2nk-2$. Secondly, assume that the starting position of $s'_{i,j}$ and the starting position of its closest front tag in $c_{i,j}$ differ by $p > 3nk$ positions. Then, a block of $3nk$ symbols "1" falls onto the encoding and/or the back tag part of $s'_{i,j}$. Since the encoding part and back tag contain together only $2+(nk-2k+2) < nk$ (under the assumption that $k>2$) many symbols "1", we have more than $2nk$ mismatching symbols and $d_H(s'_{i,j},t') > 2(nk-k)$.

Summarizing, we conclude that $s'_{i,j}$ coincides with a front tag in choice string $c'_{i,j}$, i.e., $s'_{i,j} = t' = s' = \langle \text{front\_tag} \rangle$. ∎

Lemma 4. *The encoding part of s contains exactly k symbols "1".*

Proof. Assume that s has less than k symbols "1" in its encoding part, i.e., s'' contains less than k symbols "1". Then, because $t'' = 1^{nk}$, $d_H(s'',t'') \geq nk-k+1$, implying $d_H(s,t) \geq nk-k+1$, a contradiction.

Assume that s has more than k "1" symbols in its encoding part s''. Then, $d_H(s'',s''_{i,j}) > k-2$ for the encoding part $s''_{i,j}$ of a match in every choice string $c_{i,j}$. Now consider the solution's back tag s'''. To achieve $d_H(s,s_{i,j}) \leq nk-k$, we need $d_H(s''',s'''_{i,j}) < nk-2k+2$ and s''' must contain one or more symbols "1". Every "1" symbol in s''' will decrease the value $d_H(s,s_{i,j})$ for a block $s_{i,j}$ of *one* choice string $c_{i,j}$ by one, but will increase the solution's Hamming distance to the selected blocks of *all other* choice strings. No matter how many "1" symbols we have in the back tag of s, there will always be a choice string $c_{i,j}$ with $d_H(s''',s'''_{i,j}) \geq nk-2k+2$. In summary, we will always have a choice string $c_{i,j}$ with $d_H(s,s_{i,j}) = d_H(s'',s''_{i,j}) + d_H(s''',s'''_{i,j}) > nk-k$, a contradiction. ∎

Lemma 5. *Every section of the encoding part of s contains exactly one symbol "1".*

Proof. Assume that not every section in the encoding part of s contains exactly one "1" symbol. Then, there must be a section containing no symbol "1", since, by Lemma 4, the number of symbols "1" in the encoding part of s adds up to k. Let i', $1 \leq i' \leq k$, be the section containing no symbol "1". W.l.o.g., consider a choice string $c_{i',j}$, $i' < j \leq k$ (if $i' = k$ then we consider a choice string $c_{j,i'}$, $1 \leq j < i'$ instead). In *every* block $s_{i',j}$ of $c_{i',j}$, sections i' and j of the encoding part are active and, therefore, contain exactly one symbol "1" each; these are the only symbols "1" in $s''_{i',j}$. Now consider the k symbols "1" in the encoding part of s: The "1"s in all sections of s'' except for section j are all aligned with "0"s in $s''_{i',j}$; within section j, only a single "1" of s'' can be matched to a "1" of $s''_{i',j}$. Therefore, $d_H(s'',s''_{i',j}) > k-2$. As in the proof of Lemma 4, we conclude that s is no solution. ∎

Proposition 4. *The k symbols "1" in the solution string's encoding part correspond to a k-clique in the graph.*

Proof. Let s be a solution for the *Closest Substring* instance. Summarizing, we know by Lemma 3 that s can have as a match only one of the choice string's blocks. By Lemma 5, every section of the encoding part s'' contains exactly one "1" symbol; therefore, we can read this as an encoding of k vertices of the graph. Let $v_{h_1}, v_{h_2}, \ldots, v_{h_k}$ be these vertices. Further, we know that the back tag s''' consists only of "0" symbols: By Lemma 4, the encoding part s' has only k "1"s; would s''' contain a "1", then we would have $d_H(s,t) > nk-k$. We have $d_H(s''', s'''_{i,j}) = nk-2k+2$ for *every* choice string match $s_{i,j}$ and, since every $s''_{i,j}$ contains only two "1" symbols, $d_H(s'', s''_{i,j}) \geq k-2$. Now consider some $1 \leq i < j \leq k$ and the corresponding choice string $c_{i,j}$. Since s is a solution, we know that there is a block $s_{i,j}$ with $d_H(s'', s''_{i,j}) = k-2$. That means that the two "1" symbols in $s''_{i,j}$ have to match two "1" symbols in s''; this implies that the two vertices v_{h_i} and v_{h_j} are connected by an edge in the graph. Since this is true for all $1 \leq i < j \leq k$, vertices $v_{h_1}, \ldots, v_{h_k}$ are pairwisely interconnected by edges and form a k-clique. ∎

Propositions 3 and 4 yield the following main theorem:

Theorem 2. Closest Substring *is* W[1]*-hard for parameter k in the case of a binary alphabet.*

5. *Consensus Patterns*

Our techniques for showing hardness of *Closest Substring*, parameterized by the number k of input strings, also apply to *Consensus Patterns*. Because of the similarity to *Closest Substring*, we restrict ourselves to explaining the problem and pointing out new features in the hardness proof.

Given strings $s_1, s_2, \ldots, s_k$ over alphabet Σ and integers d and L, the *Consensus Patterns* problem asks whether there is a string s of length L such that $\sum_{i=1}^{k} d_H(s, s'_i) \leq d$ where s'_i is a length L substring of s_i. Thus, *Consensus Patterns* aims for minimizing the *sum* of errors. Since errors are summed up over all strings, the value of d will, usually, not be a small and, therefore, the most significant parameterization for this problem seems to be the one by k. The problem is NP-complete and has a PTAS [25]. By reduction from *Clique*, we can show W[1]-hardness results as for *Closest Substring* given unbounded alphabet size. We omit the details here and focus on the case of binary input alphabet. We can apply basically the same ideas as were used in Section 4; however, some modifications are necessary.

5.1. Reduction of *Clique* to *Consensus Patterns*

Choice strings. As in Subsection 4.1, we generate a set of $\binom{k}{2}$ *choice strings* $S_c = \{c_{1,2}, c_{1,2}, \ldots, c_{k-1,k}\}$ with $c_{i,j} := \langle \text{block}(i,j,e_1)\rangle\langle \text{block}(i,j,e_2)\rangle \ldots \langle \text{block}(i,j,e_m)\rangle$, encoding the m edges of the input graph. This time, however, every block consists only of a front tag and an encoding part. No back tag is necessary. Therefore, we use $\langle \text{block}(i,j,(v_r,v_s))\rangle := \langle \text{front\_tag}\rangle \langle \text{encode}(i,j,(v_r,v_s))\rangle$, in which the encoding part $\langle \text{encode}(i,j,(v_r,v_s))\rangle$ is constructed as in Subsection 4.1. Before we explain the front tags, we already fix the distance value d.

Distance Value. We set the distance value $d := (\binom{k}{2} - (k-1))nk$.

Front tags. The front tag is now given by $(1^{nk^3}0)^{nk^3}0^{nk^3}$. Thus, the front tag has length $n^2k^6+2nk^3$. The front tag here is more complex than the one used in Subsection 4.1. The reason is as follows. Its purpose is to make sure that a substring which is not a block cannot be a match. To achieve this, the front tag lets such an unwanted substring necessarily have a distance value larger than d to a possible solution (as explained in the proof of Lemma 3). Since d has a higher value here compared to Section 4, we need the more complex front tag.

Solution length. We set the substring length to the length of one block, i.e., the sum of $n^2k^6+2nk^3$ (the length of the front tag) and nk (the length of the encoding part). Therefore, $L := n^2k^6 + 2nk^3 + nk$.

Template strings. In contrast to Subsection 4.1, we produce not only one but $\binom{k}{2} - (k-1)$ many template strings. All template strings have length L, i.e., the length of one block. The template strings are a concatenation of the front tag part (as given above) and an encoding part consisting of nk many symbols "1".

In summary, the front tag ensures that only the block of a choice string can be selected as a substring matching a solution. Regarding the distribution of mismatches, we note that a closest substring's front tag part will not cause any mismatches. In its encoding part, every of its nk positions causes at least $\binom{k}{2} - (k-1)$ mismatches. It causes exactly $\binom{k}{2} - (k-1)$ mismatches for every position iff the input graph contains a k-clique.

5.2. Correctness of the Reduction

Proposition 5. *For a graph with a k-clique, the construction in Subsection 5.1 produces an instance of* Consensus Patterns *which has a solution,*

i.e., there is a string s of length L such that every $c_{i,j}$, $1 \le i < j \le k$, has a substring $s_{i,j}$ with $\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} d_H(s, s_{i,j}) \le d$.

Proof. Given an undirected graph G with n vertices and m edges, let $1 \le h_1 < h_2 < \ldots < h_k \le n$ be the indices of k-clique's vertices. Then, let string s consist of the front tag described in the above construction, concatenated with the encoding part $\langle \text{number}(h_1) \rangle \langle \text{number}(h_2) \rangle \ldots \langle \text{number}(h_k) \rangle$, which encodes all clique vertices. For every $1 \le i < j \le k$, we choose in choice string $c_{i,j}$ the block $s_{i,j}$ encoding the edge connecting vertices v_{h_i} and v_{h_j}. We will show that these blocks have exactly total Hamming distance $(\binom{k}{2} - (k-1))nk$ to s.

The front tags of s and of each $s_{i,j}$ coincide, their Hamming distance is 0. Recall from Subsection 4.1 that the encoding parts consist of k sections, each section of length n. We consider the encoding parts section by section and, within a section, columnwise. Given a section i', $1 \le i' \le k$, there are $k-1$ choice strings in which this section is active, and this section in these blocks encodes vertex $v_{h_{i'}}$. Consider the column at position $h_{i'}$ in this section, over all selected substrings and all template strings. We have $\binom{k}{2} - (k-1)$ "0" symbols from the choice strings in which this section is inactive; in all other strings, there is a "1" at this position. In s, this position is "1," causing $\binom{k}{2} - (k-1)$ mismatches. Now consider the remaining columns of section i'. In each of them, we have $\binom{k}{2} - (k-1)$ "1" symbols from the template strings; all $\binom{k}{2}$ choice strings have "0" at the corresponding position. In s, this position is "0," causing $\binom{k}{2} - (k-1)$ mismatches. Thus, we have $\binom{k}{2} - (k-1)$ mismatches at every of the n positions within a section, and this is true for all k sections of the encoding part. The sum of distances from s to the matches in choice strings and the template strings is $(\binom{k}{2} - (k-1))kn$; s is a solution. ∎

For the reverse direction, we use two lemmas to show important properties that a solution of the constructed instance has. The first lemma is proved in analogy to Lemma 3.

Lemma 6. *A solution s and all its matches in the input instance start with the front tag.* □

The second property of a solution, although also valid for the solutions in Subsection 4.2, is established in a different way here. It relies on the additional template strings that have been introduced in the construction of the *Consensus Patterns* instance.

Lemma 7. *A solution s contains exactly one symbol "1" in every section of its encoding part.*

Proof. Let s be a solution for the constructed *Consensus Patterns* instance. By Lemma 6, we know that s and all its matches in the choice strings start with the front tag. Consequently, the matches in the choice strings must be blocks.

Consider the encoding part of a solution s together with the encoding parts of its matches in the input strings. We note that we have at least $\binom{k}{2} - (k-1)$ mismatches for every column at positions p, $1 \leq p \leq nk$: On the one hand, all $\binom{k}{2} - (k-1)$ template strings have "1" symbols at position p. On the other hand, all $\binom{k}{2} - (k-1)$ choice strings in which position p's section is inactive have "0" at this position, no matter which blocks we chose in these choice strings. Since s is a solution and only a total of $(\binom{k}{2} - (k-1))nk$ mismatches are allowed, we have *exactly* $\binom{k}{2} - (k-1)$ mismatches for every position of the encoding part of s with the corresponding positions in the matches of s.

Now, consider an arbitrary section i', $1 \leq i' \leq k$, and consider all $k-1$ choice strings in which section i' is active. In these choice strings, section i' contains exactly one "1" symbol. We will show that in these choice strings' blocks that form the matches for s, the "1" in section i' must be at the same position in all matches, because, otherwise, s is no solution. Assume that we chose blocks in which the "1" symbols of section i' are at different positions. We can easily check that this would cause more than $\binom{k}{2} - (k-1)$ mismatches for the columns corresponding to the positions of the "1" symbols; this would contradict the assumption that s is a solution. We conclude that, for all matches in choice strings, the "1" symbols of section i' must be at the same position. For columns in which we have "1" symbols in choice strings, there is a majority of "1" symbols, namely those in the $(k-1)$ choice strings in which section i' is active and those in the $\binom{k}{2} - (k-1)$ template strings. Therefore, the respective position in s must be "1." For all other columns, there is a majority of "0" symbols, namely those in all $\binom{k}{2}$ choice strings. Therefore, the respective position in s must be "0." ∎

These two lemmas allow us to show that also the reverse direction of the reduction is correct.

Proposition 6. *The k symbols "1" in the solution string's encoding part correspond to a k-clique in the graph.*

Proof. Let s be a solution for the constructed *Consensus Patterns* instance. By Lemma 7, every section in the encoding part of s encodes a vertex of the input graph. In the following, we show that all encoded vertices are interconnected by edges.

Let $V_C = \{v_{h_1}, v_{h_2}, \ldots, v_{h_k}\}$ be the vertices encoded in the solution's encoding part. For every two sections $1 \leq i < j \leq k$, we select in choice string $c_{i,j}$ a substring in which the "1" symbols of sections i and j are at the same positions as the "1" symbols of sections i and j in the solution: Selecting another substring would result in a Hamming distance greater than $\binom{k}{2} - (k-1)$ in the h_ith and h_jth column and s could not be a solution. Hence, the selected block encodes the edge connecting v_{h_i} and v_{h_j}. Since we find such a substring for every $1 \leq i < j \leq k$, every pair of vertices in V_C is connected by an edge, and V_C is a k-clique. ∎

Propositions 5 and 6 yield the following main result.

Theorem 3. Consensus Patterns *is* W[1]*-hard for parameter* k *in case of a binary alphabet.*

6. Conclusion

We have proven that *Closest Substring* and *Consensus Patterns*, parameterized by the number k of input strings and with alphabet size two, are W[1]-hard. This contrasts with related sequence analysis problems, such as *Longest Common Subsequence* [3,4] and *Shortest Common Supersequence* [22], where, until now, parameterized hardness has only been established in the case of unbounded alphabet size. Now, it is also known that these problems, parameterized by the number of input strings, are W[1]-hard in case of bounded alphabet size [34]. In our opinion, however, intuitively speaking, our W[1]-hardness result for *Consensus Patterns* is the most surprising one in this context, because *Consensus Patterns* seems to carry significantly less combinatorial structure than the other problems.

The parameterized complexity of *Closest Substring* and *Consensus Patterns*, parameterized by "distance parameter" d, remains open for alphabets of constant size. If these problems are also W[1]-hard, then an efficient and practically useful PTAS would appear to be impossible [6,12], unless further structure of natural input distributions is taken into account in a more complex aggregate parameterization of these basic computational string problems.

Notably, the constructions presented in this work led to a W[1]-hardness result for the slightly more general *Distinguishing Substring Selection* problem, which holds for both natural parameterizations, i.e., "number of input strings" and "distance" [19]. It is to be expected that our constructions might be useful in further hardness proofs concerning string problems.

Note added in proof. Very recently, Marx [27] stated a proof of W[1]-hardness for *Closest Substring* with respect to distance parameter d, thus answering a central open question posed in this paper.

References

[1] J. Alber, J. Gramm and R. Niedermeier: Faster exact solutions for hard problems: a parameterized point of view, *Discrete Mathematics* **229(1–3)** (2001), 3–27.

[2] M. Blanchette, B. Schwikowski and M. Tompa: Algorithms for phylogenetic footprinting, *Journal of Computational Biology* **9(2)** (2002), 211–224.

[3] H. L. Bodlaender, R. G. Downey, M. R. Fellows and H. T. Wareham: The parameterized complexity of sequence alignment and consensus, *Theoretical Computer Science* **147** (1995), 31–54.

[4] H. L. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett and H. T. Wareham: Parameterized complexity analysis in computational biology, *Computer Applications in the Biosciences* **11** (1995), 49–57.

[5] J. Buhler and M. Tompa: Finding motifs using random projections, *Journal of Computational Biology* **9(2)** (2002), 225–242.

[6] M. Cesati and L. Trevisan: On the efficiency of polynomial time approximation schemes, *Information Processing Letters* **64(4)** (1997), 165–171.

[7] J. Chen, I. Kanj and W. Jia: Vertex Cover: further observations and further improvements, *Journal of Algorithms* **41(2)** (2001), 280–301.

[8] D. Coppersmith and S. Winograd: Matrix multiplication via arithmetical progression, *Journal of Symbolic Computations* **9** (1990), 251–280.

[9] X. Deng, G. Li, Z. Li, B. Ma and L. Wang: Genetic Design of Drug without Side Effects, *SIAM Journal on Computing* **32(4)** (2003), 1073–1090.

[10] R. G. Downey: Parameterized complexity for the skeptic, in *Proc. of the 18th Annual IEEE Conference on Computational Complexity (CCC)*, pages 147–168, 2003, IEEE Computer Society Press.

[11] R. G. Downey and M. R. Fellows: Fixed-parameter tractability and completeness II: On completeness for W[1], *Theoretical Computer Science* **141** (1995), 109–131.

[12] R. G. Downey and M. R. Fellows: *Parameterized Complexity*, Springer, 1999.

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

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

[15] M. R. Fellows: Parameterized complexity: the main ideas and connections to practical computing, in *Experimental Algorithmics*, volume 2547 in LNCS, pages 51–77, 2002, Springer.

[16] M. R. Fellows: Blow-ups, win/win's, and crown rules: some new directions in FPT; in *Proc. of the 29th WG*, volume 2880 in LNCS, pages 1–12, 2003, Springer.

[17] M. R. Fellows: New directions and new challenges in algorithm design and complexity, parameterized; in *Proc. of the 8th WADS*, volume 2748 in LNCS, pages 505–520, 2003, Springer.

[18] M. Frances and A. Litman: On covering problems of codes, *Theory of Computing Systems* **30** (1997), 113–119.

[19] J. Gramm, J. Guo and R. Niedermeier: On exact and approximation algorithms for distinguishing substring selection, in *Proc. of the 14th FCT*, volume 2751 in LNCS, pages 195–209, 2003, Springer. Long version to appear under the title "Parameterized intractability of Distinguishing Substring Selection" in *Theory of Computing Systems*.

[20] J. Gramm, F. Hüffner and R. Niedermeier: Closest strings, primer design, and motif search; in L. Florea *et al.* (eds), *Currents in Computational Molecular Biology 2002*, poster abstracts of RECOMB 2002, pages 74–75.

[21] J. Gramm, R. Niedermeier and P. Rossmanith: Fixed-parameter algorithms for Closest String and related problems, *Algorithmica* **37(1)** (2003), 25–42.

[22] M. T. Hallett: *An Integrated Complexity Analysis of Problems from Computational Biology*, PhD Thesis, University of Victoria, Canada, 1996.

[23] J. K. Lanctot, M. Li, B. Ma, S. Wang and L. Zhang: Distinguishing String Search Problems, *Information and Computation* **185** (2003), 41–55.

[24] M. Li, B. Ma and L. Wang: Finding similar regions in many strings, in *Proc. of 31st ACM STOC*, pages 473–482, 1999, ACM Press. Preliminary version of [25] and [26].

[25] M. Li, B. Ma and L. Wang: Finding similar regions in many sequences, *Journal of Computer and System Sciences* **65(1)** (2002), 73–96.

[26] M. Li, B. Ma and L. Wang: On the Closest String and Substring Problems, *Journal of the ACM* **49(2)** (2002), 157–171.

[27] D. Marx: The Closest Substring problem with small distances, in *Proc. of the 46th FOCS*, pages 63–72, 2005, IEEE Press.

[28] J. Nešetřil and S. Poljak: On the complexity of the subgraph problem, *Commentationes Mathematicae Universitatis Carolinae* **26(2)** (1985), 415–419.

[29] R. Niedermeier: Ubiquitous parameterization – invitation to fixed-parameter algorithms, in *Proc. of 29th MFCS*, volume 3153 in LNCS, pages 84–103, 2004, Springer.

[30] R. Niedermeier: *Invitation to Fixed-Parameter Algorithms*, Oxford University Press, 2006.

[31] R. Niedermeier and P. Rossmanith: Upper bounds for Vertex Cover further improved, in *Proc. of 16th STACS*, volume 1563 in LNCS, pages 561–570, 1999, Springer.

[32] P. A. Pevzner: *Computational Molecular Biology – An Algorithmic Approach*, The MIT Press, 2000.

[33] P. A. Pevzner and S.-H. Sze: Combinatorial approaches to finding subtle signals in DNA sequences, in *Proc. of 8th ISMB*, pages 269–278, 2000, AAAI Press.

[34] K. Pietrzak: On the parameterized complexity of the fixed alphabet Shortest Common Supersequence and Longest Common Subsequence problems, *Journal of Computer and System Sciences* **67(1)** (2003), 757–771.

[35] M.-F. Sagot: Spelling approximate repeated or common motifs using a suffix tree, in *Proc. of the 3rd LATIN*, volume 1380 in LNCS, pages 111–127, 1998, Springer.

Michael R. Fellows

Department of Computer Science
and Software Engineering
University of Newcastle
University Drive
Callaghan 2308
Australia
mfellows@cs.newcastle.edu.au

Rolf Niedermeier

Institut für Informatik
Universität Jena
Ernst-Abbe-Platz 1–4
D-07740 Jena
Germany
niedermr@minet.uni-jena.de

Jens Gramm

Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Sand 13
D-72076 Tübingen
Germany
gramm@informatik.uni-tuebingen.de