Parameterized Intractability of Closest Substring

Michael R. Fellows[1], Jens Gramm[2]*, and Rolf Niedermeier[2]

[1] Department of Computer Science and Software Engineering, University of Newcastle, University Drive, Callaghan 2308, Australia
mfellows@cs.newcastle.edu.au
[2] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
{gramm, niedermr}@informatik.uni-tuebingen.de

Abstract. We show that Closest Substring, one of the most important problems in the field of biological sequence analysis, is W[1]-hard with respect to the number k of input strings (even over binary alphabet). This means that it is highly unlikely to find fixed parameter algorithms for that problem with a running time exponential only in k—according to current parameterized complexity theory the problem is fixed parameter intractable. For computational biology practice, this means that it should not be tried to "tune" existing exact algorithms with respect to parameter k. Our result also gives concrete "structural support" for the intuition that Closest Substring is much harder than the special case Closest String, although both problems are NP-complete and both possess a PTAS.
Further results show W[1]-hardness also with respect to other parameterizations in the case of unbounded alphabet size. In addition, we can extend our main W[1]-hardness result to Consensus Patterns, a problem of similar biological importance as Closest Substring is.

1 Introduction

According to Li *et al.* [11], Closest Substring "is a key open problem in search for a potential genetic drug sequence which is 'close' to some sequences (of harmful germs) and 'far' from some other sequences (of humans)" (also see [6, 10, 12] and the references cited therein for further motivation concerning applications in computational biology). Closest Substring is defined as follows.

Input: k strings $s_1, s_2, \ldots, s_k$ over alphabet Σ and integers d and L.
Question: Is there a string s of length L such that, for all $i = 1, \ldots, k$, $d_H(s, s'_i) \leq d$ where s'_i is a length L substring of s_i?

* Supported by the Deutsche Forschungsgemeinschaft (DFG), project OPAL (optimal solutions for hard problems in computational biology), NI 369/2-1.

Closest Substring is NP-complete—already the special case Closest String (where the string s we search for is of same length as the input strings) is NP-complete even in case of a binary input alphabet [7, 10]. On the positive side, both problems possess a polynomial time approximation scheme (PTAS) [11, 12]. Both these approximation algorithms impractical, however.

For Closest String as well as for Closest Substring it is natural and important to study their parameterized complexity [4]. Taking into account that the number k of strings or the distance d are comparatively small in many practical situations (cf. [6, 5, 11]), it is important to know whether the problems are fixed parameter tractable with respect to (one of) these parameters.[1] For instance, as to Closest String, very recently it was shown that it can be solved in linear time for constant d (the exponential term in d being bounded by d^d) and also in linear time for constant k; the exponential term in k, however, is huge and makes it impractical for $k > 3$ [8]. By way of contrast, the parameterized complexity of Closest Substring remained open to a large extent. Until now, it only was known that if one additionally considers the substring length L as a parameter, then exponential running times with exponent L can be achieved. Our main result is to show that Closest Substring with parameter k is W[1]-hard even for binary alphabet; that is, the problem is fixed parameter intractable unless a very unlikely collapse of the W-hierarchy (well-known in parameterized complexity [4]) occurs. For computational biologists the fundamental message is that exact algorithms for Closest Substring, parameterized by the number of strings k, probably cannot be significantly better than trivial $O(n^k)$ algorithms, where n is the total input size. For unbounded alphabet size, we can show that the problem is even W[1]-hard for parameters L, d, and k together. Note, however, that the parameterized complexity in case of constant alphabet size for the time being remains open when parameterized by d and k or d alone. We also show that also the related Consensus Pattern problem, parameterized by the number of strings, is W[1]-hard for binary alphabet.

Finally, it is worth noting that analogous parameterized complexity studies have been performed for the famous Longest Common Subsequence (LCS) problem [2, 3]. There, however, parameterized hardness results could only be shown in case of unbounded alphabet size and, in particular, it is a long-standing open problem (also cf. [4]) to determine the parameterized complexity of LCS with respect to parameter k for constant alphabet size. Constant alphabet size, however, is the case of biological importance. In this sense, our result seems to contribute the first "real" parameterized intractability result for a core problem in the field of biological sequence analysis. Eventually, 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 [8]). Notably, this could *not* be expressed by "classical complexity measures," since both problems are NP-complete as well as both do have a PTAS.

Some proofs are deferred to the Appendix.

[1] Remark that, e.g., much work has been invested to study the complexity of the similar Longest Common Subsequence problem even for $k = 2$ (see, e.g., [15]).

2 Preliminaries and Previous Work

In this section, we start with a brief introduction to parameterized complexity (details to be found in [4]) and then continue with sketching some previous work on CLOSEST STRING and related problems.

2.1 A Crash Course in Parameterized Complexity

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 $O(kn + 1.3^k)$. By way of contrast, consider the also 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 k or fewer 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 $f(k)n^{O(1)}$, where f might be an arbitrarily fast growing function only depending on k [4]. The best known algorithm solving CLIQUE runs in time $O(n^{ck/3})$, where c is the exponent on the time bound for multiplying two integer $n \times n$ matrices. The 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 [4]. Downey and Fellows developed a completeness program for showing parameterized intractability. 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 $L, L' \subseteq \Sigma^* \times \mathbf{N}$ be two parameterized languages. 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 $\mathbf{N}$ to $\mathbf{N}$ and a function $(x, k) \mapsto x'$ from $\Sigma^* \times \mathbf{N}$ 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 [4].

Now, the "lowest class of parameterized intractability", so-called $W[1]$, can be defined as the class of languages that reduce to the so-called SHORT TURING MACHINE ACCEPTANCE problem (also known as the k-STEP HALTING problem) [4]. Here, we want to determine for an input consisting of a nondeterministic Turing machine M and a string x, whether M has a computation path accepting x in at most k steps. In some sense, this is the parameterized analogue to the TURING MACHINE ACCEPTANCE problem—the basic generic NP-complete problem in

classical complexity theory. Some perhaps more natural problems being $W[1]$-*complete* are CLIQUE and INDEPENDENT SET . As a matter of fact, a whole hierarchy of parameterized intractability can be defined, $W[1]$ only being the lowest level. We omit any further details in this direction and just refer to the monograph of Downey and Fellows [4].
From a practical point of view, however, it is probably sufficient to distinguish between $W[1]$-hardness and membership in FPT, the class of fixed parameter tractable problems. Thus, for an algorithm designer not being able to show fixed-parameter tractability of a problem, it is sufficient to give a reduction from, e.g., CLIQUE to the given problem using a standard parameterized m-reduction.[2] This gives a concrete indication that, unless $P = \mathrm{NP}$, the problem is unlikely to allow for an $f(k)n^{O(1)}$ time algorithm.

2.2 CLOSEST SUBSTRING: Motivation and Previous Results

Many biological problems with respect to DNA, RNA, or protein sequences can be solved based on consensus word analysis [14, Section 8.6]; CLOSEST SUBSTRING is a central problem in this context. Applications include locating binding sites and finding conserved regions in unaligned sequences in genetic drug target identification, in designing genetic probes, and in universal PCR primer design. These problems can be regarded as various generalizations of the common substring problem, allowing errors (see [10, 11] and references there). This leads to CLOSEST SUBSTRING, where errors are modeled by the (Hamming) distance parameter d.
There is a straightforward factor 2 approximation algorithm for CLOSEST SUBSTRING. The first better-than-2 approximation with factor $2 - 2/(2|\Sigma| + 1)$ was given by Li *et al.* [11] and this was further improved to factor $4/3+\epsilon$ for any small constant $\epsilon > 0$ [10] Finally, Ma [12] presented a PTAS for CLOSEST SUBSTRING, which, however, has an impractical running time.
Concerning exact (parameterized) algorithms for CLOSEST SUBSTRING, we only briefly mention that Sagot [13] studies motif discovery algorithms, Blanchette [1] developed a so-called phylogenetic footprinting method, and Evans and Wareham [6] give FPT algorithms for CLOSEST SUBSTRING. All these results, however, make essential use of the parameter "substring length" L and show "exponential behavior with respect to L". Our analysis makes a first step towards showing that we have to include L in the exponential growth; namely, we show that is unlikely to find algorithms with a running time exponential *only* in k.

3 Unbounded Alphabet

We present a parameterized reduction from CLIQUE to CLOSEST SUBSTRING which is a parameterized one for parameters L, d, and k. This shows that CLOSEST SUBSTRING is W[1]-hard for L, d, and k with alphabet of unbounded size.

[2] We present such a (technically involved) reduction from CLIQUE to CLOSEST SUBSTRING.

3.1 Reduction from CLIQUE

A CLIQUE instance is given by a graph G, 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 clique size. We describe how to generate a set S of $\binom{k}{2}$ strings and integers $d := k-2$ and $L := k+1$ such that G has a clique of size k iff there is a string s of length L and every $s_i \in S$ has a substring s'_i of length L with $d_H(s, s'_i) \leq d$. 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 the correctness of our construction, note that we assume $k > 2$, because $k = 1, 2$ are trivial cases.

Alphabet. The alphabet of the produced instance is given by

$$\{\, \sigma_i \mid v_i \in V \,\} \dot\cup \{\, \varphi_j \mid j = 1, \ldots, \tbinom{k}{2} \,\} \dot\cup \{\#\}.$$

In words, we have a set of *encoding symbols*, i.e., an alphabet symbol for every vertex of the input graph, further we have a unique symbol for every of the $\binom{k}{2}$ produced strings, and a *synchronizing symbol* #, making a total of $n + \binom{k}{2} + 1$ alphabet symbols.

Choice strings. We generate a set of $\binom{k}{2}$ *choice strings*

$$S_c = \{c_{1,1}, c_{1,2}, \ldots, c_{1,k}, c_{2,3}, c_{2,4}, \ldots, c_{2,k}, \ldots, c_{k-1,k}\}.$$

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 single symbols that are uniquely determined for every choice string. A choice string $c_{i,j}$, which is the i'th choice string in S_c, is given by

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

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 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, $1 \leq r < s \leq n$. Then, the ith position of the block is σ_r to encode v_r and the jth position is σ_s to encode v_s. The last position of a block is set to the synchronizing symbol #. Let $c_{i,j}$ be the i'th choice string in S_c; then, all remaining positions in the block are set to character $\varphi_{i'}$, which is unique for the choice string. Thus, the block is given by

$$\langle \text{block}(i, j, (v_r, v_s)) \rangle \;:=\; \varphi_{i',1}\, \varphi_{i',2} \cdots \varphi_{i',i-1}\, \sigma_r\, \varphi_{i',i+1} \cdots \varphi_{i',j-1}\, \sigma_s\, \varphi_{i',j+1} \cdots \varphi_{i',k}\, \#,$$

where all $\varphi_{i',1}, \varphi_{i',2}, \ldots, \varphi_{i',k}$ stand for $\varphi_{i'}$.

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

3.2 Correctness of the Reduction

In the following proposition, we show that a k-clique in the input graph implies a Closest Substring instance with a solution. Afterwards, we show the more difficult reverse direction.

Proposition 1 *For a graph with a k-clique, the construction in Subsection 3.1 produces a Closest Substring with 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 given by $s := \sigma_{h_1}\sigma_{h_2}\ldots\sigma_{h_k}\#$. 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}. By the construction, 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:

$$s_{i,j} := \varphi_{i',1}\,\varphi_{i',2}\ldots\varphi_{i',i-1}\,\sigma_{c_i}\,\varphi_{i',i+1}\ldots\varphi_{i',j-1}\,\sigma_{c_j}\,\varphi_{i',j+1}\ldots\varphi_{i',k}\,\#,$$

where i' is the number of the choice string in S_c. We have $d_H(s, s_{i,j}) = k - 2$, and we can find such a block for every $1 \leq i < j \leq k$. □

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. To do this, we need the following lemma (proof see Appendix).

Lemma 1 *A solution has the following properties:*

1. *A solution does not contain a symbol φ_i, $i = 1, \ldots, \binom{k}{2}$.*
2. *A solution contains exactly one # symbol, at its last position.*

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

Proof. By Lemma 1, we know that a solution 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 a solution in the choice string. Now assume that a solution is given by $s = \sigma_{h_1}\sigma_{h_2}\ldots\sigma_{h_k}\#$ 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, have # at their last position, and all other positions are set to a symbol unique for this choice string. Thus, we can only find a block that is a match if there is a block with σ_{h_i} at its ith position and σ_{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 solution 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 Note that hardness for all three parameters also implies hardness for each subset of them.

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

4 Binary Alphabet

We modify the reduction from Section 3 to achieve a Closest Substring instance with binary alphabet. The reduction is parameterized by k and, therefore, shows that Closest Substring is W[1]-hard for parameter k, even with binary alphabet. In contrast to Section 3, we are not allowed to encode every vertex with its own symbol and 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 additonal synchronizing symbol. To overcome these problems, we complement the set of produced strings by one addtional string and lengthen the blocks in the produced choice strings considerably.

4.1 Reduction from Clique

Number strings. To encode numbers between 1 and n, we introduce *number strings* $\langle \text{number}(pos) \rangle$, which have length n and have a "1" symbol at position pos and "0" symbols elsewhere:

$$\langle \text{number}(pos) \rangle \ := \ 0_1 0_2 \dots 0_{pos-1} \, 1 \, 0_{pos+1} \dots 0_n,$$

where all $0_1, 0_2, \dots, 0_n$ stand for the "0" symbol. In contrast to the reduction from Section 3, we can now use these number strings to replace the symbols encoding the vertices of a graph.

Choice strings. As in Section 3, we generate a set of $\binom{k}{2}$ *choice strings*

$$S_c = \{c_{1,1}, c_{1,2}, \dots, c_{1,k}, c_{2,3}, c_{2,4}, \dots, c_{2,k}, \dots, c_{k-1,k}\}.$$

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

$$c_{i,j} \ := \ \langle \text{block}(i,j,e_1) \rangle \langle \text{block}(i,j,e_2) \rangle \langle \text{block}(i,j,e_3) \rangle \dots \langle \text{block}(i,j,e_m) \rangle,$$

where $e_1, e_2, \dots, e_m$ are the edges of the input graph and $\langle \text{block}() \rangle$ will be defined below. The length of the solution string 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 i'th string in S_c. Then, the 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 solution string can only match substrings at certain positions in the produced choice strings, using front tags:

$$\langle \text{front\_tag} \rangle \ := \ (1^{3nk}0)^{2nk},$$

i.e., a front tag has length $(3nk+1) \cdot 2nk$. By this arrangement, the solution s and every match of s start (see Subsection 4.2) with the front tag.

Encoding part. The encoding part consists of k sections, each section has length n. The encoding part corresponds to the blocks used in Section 3. Consequently, in $\langle \text{block}(i,j,e)\rangle$ the ith and jth section are called *active* and encode edge $e = (v_r, v_s)$; section i encodes v_r and section j encodes v_s. The other sections except sections i and j are called *inactive*. Thus,

$$\begin{aligned}\langle \text{encode}(i,j,(v_r,v_s))\rangle := \langle \text{inactive}_1\rangle \ldots \langle \text{inactive}_{i-1}\rangle \langle \text{active}(r)\rangle \langle \text{inactive}_{i+1}\rangle \\ \ldots \langle \text{inactive}_{j-1}\rangle \langle \text{active}(s)\rangle \langle \text{inactive}_{j+1}\rangle \ldots \langle \text{inactive}_k\rangle.\end{aligned}$$

The sections $\langle \text{inactive}\rangle$ are simply length n strings of "0" symbols. An active section $\langle \text{active}(v_r)\rangle$, encoding the rth vertex from the set of n vertices, is given by a number string of length n, encoding number r: $\langle \text{active}(r)\rangle := \langle \text{number}(r)\rangle$

Back tag. The back tag of a block is intended to balance the Hamming distance of the solution string 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 "1" symbols, all other sections consist of "0" symbols:

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

Thus, the back tag of a block has length $\binom{k}{2}(nk-2k+2)$.

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 $(3nk+1)\cdot 2nk$, followed by a length nk string of "1" symbols, followed by a length $nk-2k+2$ string of "0" symbols. Thus, the template string has the same length as a block in a choice string, i.e, $(3nk+1)\cdot 2nk + nk + \binom{k}{2}(nk-2k+2)$.

Values for d and L. We set $L := (3nk+1)\cdot 2nk + nk + \binom{k}{2}(nk-2k+2)$ and $d := nk - k$. As we will see 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 solution string s, we denote its first $(3nk+1)\cdot 2nk$ symbols, the front tag, by s', the following nk symbols, its encoding part, by s'', and the last $\binom{k}{2}(nk-2k+1)$ 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 $s'''_{i,j}$.

4.2 Correctness of the Reduction

First, we show that a k-clique in the input graph results in a Closest Substring instance having a solution and, secondly, that a solution for the Closest Substring instance implies a k-clique.

Proposition 3 *For a graph with a k-clique, the outlined construction produces a* Closest Substring *instance with 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 $d_H(s, s_{i,j}) \le d$ and $d_H(s,t) \le d$.*

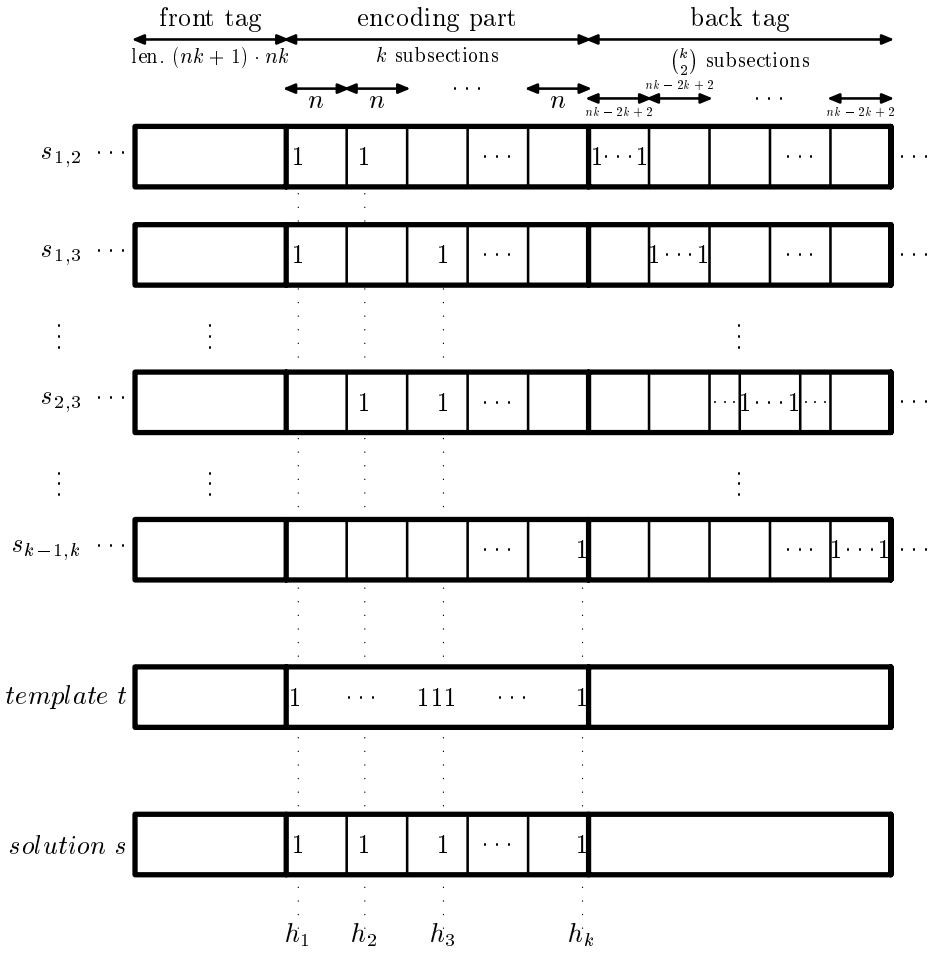

Fig. 1. Illustration of the solution for the Closest Substring instance that is constructed from a graph having a k-clique. We display the matches in the choice strings, the template string and the solution string. The front tags are equal in all strings and, therefore are not displayed in detail. For the encoding part, we only indicate the positions of "1" symbols, where $h_1, h_2, \ldots, h_k$ name the position of a "1" symbol within its section. For the back tag part, we also only indicate the sections which consist of "1" symbols, all other sections consist of "0" symbols.

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 solution as outlined in Fig. 1. We display the blocks in the choice strings that are chosen as substrings, the template string, and the solution string. It follows from the construction that the choice strings have the indicated substrings: For every $1 \leq i < j \leq k$, it produced choice string $c_{i,j}$ with a block $s_{i,j}$ encoding the edge

between vertices v_{h_i} and v_{h_j}. This block is a match for our solution. For this kind of block of a choice string as well as for the template string, we report the distance they have to the solution string, separately for each of their two parts and in total:

	s'	s''	s'''	s
choice string $s_{i,j}$	$d_H(s', s'_{i,j}) = 0$	$d_H(s'', s''_{i,j}) = k-2$	$d_H(s''', s'''_{i,j}) = nk-2k+2$	$d_H(s, s_{i,j}) = nk-k$
template string t	$d_H(s', t') = 0$	$d_H(s'', t'') = nk-k$	$d_H(s''', t''') = 0$	$d_H(s, t) = nk-k$

As is obvious from these distance values, the indicated substrings in the choice strings all have Hamming distance $nk-k$ to the solution string. □

For the reverse direction, we assume that the CLOSEST SUBSTRING instance has a solution. We need the following statements (proof see Appendix):

Lemma 2 *A solution has the following properties:*

1. *A solution and all its matches in the input instance start with the front tag.*
2. *A solution's encoding part contains at least k "1" symbols.*
3. *A solution's encoding part contains at most k "1" symbols.*
4. *Every section of a solution's encoding part contains exactly one "1".*

Lemma 2 directly implies that the only matches in a choice string $c_{i,j}$ are its blocks and that s''' consists only of "0" symbols.

Proposition 4 *The k "1" symbols 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 2(4) that, in the encoding part s'', every section 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 2(1), we have a solution only if we match one of the choice string's blocks. We have $d_H(s''', s'''_{i,j}) = nk-2k+2$ for *every* choice string $c_{i,j}$ 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'' 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 $s''_{i,j}$; 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 $v_1, \ldots, v_k$ are 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 already for binary alphabet.*

5 The Consensus Patterns Problem

Our ideas for showing parameterized intractability of Closest Substring, parameterized by the number k of input strings, also apply to the related Consensus Patterns problem. Due to similarity of Closest Substring, we restrict to explaining the problem and pointing out new features in the reduction.
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 $\sum_{i=1,\ldots,k} d_H(s, s'_i) \leq d$ where s'_i is a length L substring of s_i?
The problem is NP-complete and has a PTAS [11]. By reduction from Clique, we can show W[1]-hardness in analogy to Section 3. We omit the details here and focus on the case of binary alphabet. We can apply, the same ideas as they were used in Section 4; however, we need some modifications. We outline the differences of the reduction in the following.
As in Subsection 4.1, we generate a set of $\binom{k}{2}$ choice strings $c_{1,1}, \ldots, ck-1, k$; each choice string consists of m blocks, one block for every edge of the input graph. The blocks, however, do consist of only two parts, the front tag and the encoding part. The back tag part is omitted. The encoding part is constructed as in Subsection 4.1. The front tag, however, is now given by $0^x(1^x0)0^x$ where $x := (\binom{k}{2}(k-1))nk$. We set paremeter L to the block length: $L := (\binom{k}{2}(k-1))^2 + ((4\binom{k}{2}(k-1))+1)nk+1$. In contrast to Subsection 4.1, we produce not only one but $\binom{k}{2} - (k-1)$ many template strings. All template strings are equal, have length L, and are a concatenation of the front tag part (as given above) and an encoding part consisting only of "1" symbols. The distance parameter is set to $d := (\binom{k}{2} - (k-1))nk$.
For an overview, 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 solution's front tag part won't 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. Based on this reduction, we state the following theorem (proof in the appendix).

Theorem 3 Consensus Patterns *is W[1]-hard for parameter k already for binary alphabet.*

6 Conclusion and Open Questions

Proving that Closest Substring, parameterized by the number k of input strings and with alphabet size two, is W[1]-hard, we gave the seemingly first parameterized intractability result for a core problem from biological sequence analysis. This stands in contrast to related problems such as Longest Common Subsequence [2, 3] and Shortest Common Supersequence [9], where parameterized hardness could only be shown in case of unbounded alphabet size.

Many directions can be pursued in future research. For instance, the parameterized complexity of Closest Substring, parameterized by "distance parameter" d, remains open. Also, it would be interesting to study these problems when using edit distance instead of Hamming distance as a measure. Finally, we leave it as an open problem whether Closest Substring, parameterized by k, is contained in W[1] and, thus, is W[1]-complete. (It seems easy to see that it is contained in W[P].

References

1. M. Blanchette. Algorithms for phylogenetic footprinting. In *Proc. of 5th ACM RECOMB*, pages 49–58, 2001, ACM Press.
2. H. L. Bodlaender, R. G. Downey, M. R. Fellows, and H. T. Wareham. The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science*, 147:31–54, 1995.
3. 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: 49–57, 1995.
4. R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer. 1999.
5. P. A. Evans and H. T. Wareham. Practical algorithms for universal DNA primer design: an exercise in algorithm engineering. In N. El-Mabrouk *et al.* (eds.) *Currents in Computational Molecular Biology 2001*, pages 25–26, Les Publications CRM, 2001.
6. P. A. Evans and H. T. Wareham. Practical non-polynomial time algorithms for designing universal DNA oligonucleotides: a systematic approach. Manuscript, April 2001.
7. M. Frances and A. Litman. On covering problems of codes. *Theory of Computing Systems*, 30:113–119, 1997.
8. J. Gramm, R. Niedermeier, and P. Rossmanith. Exact solutions for Closest String and related problems. Accepted for the *12th ISAAC*, December 2001. To appear in LNCS, Springer.
9. M. T. Hallett. *An Integrated Complexity Analysis of Problems from Computational Biology.* PhD Thesis, University of Victoria, Canada, 1996.
10. J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. In *Proc. of 10th ACM-SIAM SODA*, pages 633–642, 1999, ACM Press. To appear in *Information and Computation.*
11. 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.
12. B. Ma. A polynomial time approximation scheme for the closest substring problem. In *Proc. of 11th CPM*, number 1848 in LNCS, pages 99–107, 2000. Springer.
13. M.-F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *Proc. of 3rd LATIN*, number 1380 in LNCS, pages 111-127, 1998. Springer.
14. Pavel A. Pevzner. *Computational Molecular Biology - An Algorithmic Approach.* MIT Press. 2000.
15. D. Sankoff and J. Kruskal (eds.). *Time Warps, String Edits, and Macromolecules.* Addison-Wesley. 1983. Reprinted in 1999 by CSLI Publications.

A Proofs omitted in the Text

Proof of Lemma 1 from Subsection 3.2:
(1) A solution does not contain a symbol φ_i, $i = 1, \ldots, \binom{k}{2}$. Let S_φ be the set of symbols from $\{ \varphi_i \mid 1 \leq i \leq \binom{k}{2} \}$ that occur in the solution. Obviously, $|S_\varphi| \leq k+1$. Assume that $|S_\varphi| > 0$.
It is easy to observe that $|S_\varphi| \leq k-2$: Otherwise, there would be (at least) $\binom{k}{2} - (k+1)$ choice strings which contain no symbol of S_φ since every such symbol is unique for one choice string. In these strings, any length $k+1$ substring would have Hamming distance more than $k-2$ to the solution.
With $|S_\varphi| \leq k-2$, there are at least $\binom{k}{2} - (k-2)$ choice strings which contain no symbol of S_φ. Every length $k+1$ substring of these strings contains at most two encoding symbols and at most one # symbol. To achieve Hamming distance $k-2$ to the solution, we must choose a length $k+1$ substring with two encoding symbols and one # symbol that all match the corresponding symbols in the solution. Having one # and the symbols from S_φ, $|S_\varphi| > 0$, in the solution, there are at most $k-1$ encoding symbols. The # symbol ensures that all matches start at the same position relative to a # symbol. Since every choice string has a different pair of active positions, the $k-1$ encoding symbols can provide matches for at most $\binom{k-1}{2}$ choice strings. But $\binom{k-1}{2} < \binom{k}{2} - (k-2)$ such that there are choice strings which have no substring with Hamming distance at most $k-2$ to our assumed solution. This shows that the assumed $k+1$ string with $0 < |S_\varphi|$ is no solution and conclude that $|S_\varphi| = 0$. □
(2) A solution contains exactly one # symbol, at its last position. By part 1 of this lemma, we know that the solution can only contain encoding symbols and the synchronizing symbol. Firstly, assume that the solution does not contain exactly one synchronizing symbol. Since we can easily check that every length $k+1$ substring of a choice string has at most two encoding symbols and at most one synchronizing symbol, there would be a choice string with Hamming distance greater than $k-2$ to the assumed solution, a contradiction.
Secondly, assume that a solution has the synchronizing symbol not at its last position but at position p, $1 \leq p \leq k$. Note that a match must contain the synchronizing symbol also at position p. Then, let $p' := k - p + 1$ and consider, e.g., choice string $c_{1,p'}$. In its blocks only symbols at position 1 and p' within one block are active and they are set to encoding symbols. Those length $k+1$ substrings with the synchronizing symbol at their pth position, however, do not include the p'th position of any block in the choice string. Therefore, they have Hamming distance greater than $k-2$ to the assumed solution, a contradiction. □

Proof of Lemma 2 from Subsection 4.2:
(1) A solution and all its matches in the input instance start with the front tag. Let s be a solution. Since s is of length $L = (3nk+1) \cdot 2nk + nk + \binom{k}{2}(nk-k-2)$, 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

possible substring in a choice string $c_{i,j}$ that can match with at most d differences is 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 $(3nk+1) \cdot 2nk$ 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.

First, assume that the starting position of $s'_{i,j}$ and the starting position of a front tag in $c_{i,j}$ differ by $3nk$ or less positions. Then, at least $2nk-1$ "0" symbols of t' are matched with "1" symbols of the front tag in $s'_{i,j}$ and $d_H(s'_{i,j}, t') > 2(nk-k)$. Secondly, assume that $s'_{i,j}$ is shifted by more than $3nk$ positions related to the position of a front tag. Then a block of $3nk$ "1" symbols falls onto the encoding and/or the back tag part of $s'_{i,j}$. Since the encoding part and back tag contain together only $k + nk - 2k + 1 < nk$ "1" symbols, 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., $s'_{i,j} = t' = s' = \langle \text{front\_tag} \rangle$. □

(2) A solution's encoding part contains at least k "1" symbols. Assume that a solution s has less than k "1" symbols in its encoding part, i.e., s'' contains less than k "1"s. Then, $d_H(s'', t'') = nk - k + 1$. Therefore, $d_H(s, t) \geq nk - k + 1$, a contradiction. □

(3) A solution's encoding part contains at most k "1" symbols. Assume that a solution 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 every choice string $c_{i,j}$. Now consider the solution's back tag s'''. Assume that it contains one or more "1" symbols. Every "1" symbol 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, there will always be a choice string $c_{i,j}$ with $d_H(s''', s'''_{i,j}) \geq nk - 2k + 2$. In total, we will always have a choice string $c_{i,j}$ with $d_H(s, s_{i,j}) = d_H(s'', S''_{i,j}) + d_H(s''', S'''_{i,j}) > nk - k$. □

(4) Every section of a solution's encoding part contains exactly one "1". Let s be a solution and assume that not every section its encoding part contains exactly one "1" symbol. Then, there must be a section containing no "1" symbol, since, by part 3 of this lemma, the number of "1" symbols in the solution's encoding part adds up to k. Let i', $1 \leq i' \leq k$, be the section containing no "1" symbol. Now consider a choice string $c_{i',j}$, $1 \leq i' < j \leq k$. In every block $s_{i,j}$ of $c_{i',j}$, sections i' and j contain exactly one "1" symbol. At most one of the k "1" symbols in the solution's encoding part can match a "1" symbol in $s'_{i',j}$. Therefore, $d_H(s'', s''_{i,j}) > k - 2$. As in the proof of part 3, we conclude that s is no solution. □

Proof of Theorem 3 from Section 5. (Sketch)
We show that outlined construction reduces CLIQUE to CONSENSUS PATTERN.

(CLIQUE $\Rightarrow$ CONSENSUS PATTERN). We show that, if the input graph has a k-clique, the produced CONSENSUS PATTERN instance has a solution. Let $1 \leq h_1 < h_2 < \ldots < h_k \leq n$ be the indices of the k-clique's vertices. Then let a string s consist of the front tag of the construction, concatenated with the encoding part as constructed in the proof of Proposition 3. For any $1 \leq i < j \leq k$, we choose in choice string $c_{i,j}$ the block encoding the edge connecting vertices v_{h_i} and v_{h_j}. For a section $1 \leq i' \leq k$, there are $k-1$ choice strings in which this section is active, and all these sections in the selected blocks encode 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 s, this position is "1", making $\binom{k}{2} - (k-1)$ mismatches. Now consider all other columns in this section. We have $\binom{k}{2} - (k-1)$ "1" symbols from the template strings. In s, this position is "0", making $\binom{k}{2} - (k-1)$ mismatches. Thus, at every of the nk positions in the encoding part, we make $\binom{k}{2} - (k-1)$ mismatches; the sum of Hamming distances from the solution to each of the choice strings and each of the template strings is $(\binom{k}{2} - (k-1))kn$ and s is a solution for the constructed CONSENSUS PATTERN instance.

(CONSENSUS PATTERN $\Rightarrow$ CLIQUE). Let s be a solution for the constructed CONSENSUS PATTERN instance. Similar to the proof of Lemma 1, we can show that s and every selected substring has to start with the front tag. Consequently, the selected substrings of the choice strings must be blocks. Thus, this part of s will cause no mismatches.

Regarding the encoding part of s, we note that we have at least $\binom{k}{2} - (k-1)$ mismatches for every position p, $1 \leq p \leq nk$: On the one side, all $\binom{k}{2} - (k-1)$ template strings have "1" symbols at position p. On the other side, all $\binom{k}{2} - (k-1)$ choice strings in which position p's section is inactive have "0" at this position, no matter which block we chose in this choice string. Since s is a solution and only $(\binom{k}{2} - (k-1))nk$ mismatches are allowed, we have *exactly* $(\binom{k}{2} - (k-1))$ for every position of s's encoding part.

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 all selected substrings we have one "1" symbol in this section. In case we chose substrings where this section's "1" symbols are at different positions, we can easily check that this would cause more than $\binom{k}{2} - (k-1)$ mismatches for the positions with "1" symbols. Therefore, for all selected substrings, the "1" symbols of section i' must be at the same position. It also follows that every section in the solution contains exactly one "1" symbol.

With these observations we conclude that every section in s encodes a vertex of the graph. Let $V = \{v_{h_1}, v_{h_2}, \ldots, v_{h_k}\}$ be these vertices. For every two sections $1 \leq i < j \leq k$, we selected in choice string $c_{i,j}$ a substring that encodes the edge connecting v_{h_i} and v_{h_j}. Since we find such a substring for every $1 \leq i < j \leq k$, the vertices in V form a clique. □