LCNS, VOL 2285, PP. 262–273, SPRINGER 2002

On the Parameterized Intractability of CLOSEST SUBSTRING and Related Problems

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 a binary alphabet). This problem is therefore unlikely to be solvable in time $O(f(k)n^c)$ for any function f and constant c independent of k — effectively, the problem can 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 and both possess polynomial time approximation schemes. We also prove W[1]-hardness for other parameterizations in the case of unbounded alphabet size. Our main W[1]-hardness result generalizes to CONSENSUS PATTERNS, a problem of similar significance in computational biology.

1 Introduction

According to Li *et al.* [11], CLOSEST SUBSTRING "is a key open problem in the search for a potential generic drug sequence which is 'close' to some sequences (of harmful germs)..." (also see [6, 10, 12] and the references cited therein for other 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?

Herein, $d_H(s, s_i)$ denotes the Hamming distance between s and s_i. 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

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

to a binary alphabet [7,10]. On the positive side, both problems admit a polynomial time approximation scheme (PTAS) [11,12], although in both cases the exponent of the polynomial bounding the running time depends on the goodness of the approximation — they are not EPTASs in the sense of [4].

For Closest String as well as for Closest Substring it is natural and important to study their parameterized complexity [5]. Taking into account that the number k of strings or the distance d are comparatively small in many practical situations (cf. [6,11]), it is important to know whether the problems are fixed parameter tractable with respect to these parameters.[1] Closest String was recently shown to be linear time fixed parameter tractable for parameter d, for a parameter function bounded by d^d, and also linear time fixed parameter tractable for parameter k, but in this case the parametric complexity contribution is much less encouraging [8]. The parameterized complexity of Closest Substring has largely remained open.

Our main result is to show that Closest Substring with parameter k is W[1]-hard even for a binary alphabet. That is, the problem is fixed parameter intractable unless FPT = W[1], the parametric analog of P = NP.

For computational biologists, the fundamental message is that exact algorithms for Closest Substring, parameterized by the number of strings k, probably cannot achieve the running time $f(k) \cdot n^{O(1)}$ (i.e., exponential *only* in k). For unbounded alphabet size, we show that the problem is W[1]-hard for the combined parameters L, d, and k. In the case of constant alphabet size, the complexity of the problem remains open when parameterized by d and k together, or by d alone. We also show that the closely related Consensus Patterns problem [11], where one tries to minimize the *sum* of distances instead of the maximum distance, is W[1]-hard for a binary alphabet when parameterized by the number of strings. 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 still 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 [1,6,15].

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 (cf. [5]) to determine the parameterized complexity of LCS with respect to the parameter k for constant alphabet size. Constant alphabet size, however, is the case of biological importance. In this sense, our results seem to contribute some of the first "real" parameterized intractability results for core problems 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]). No-

[1] Many investigations have been made of the complexity of the similar Longest Common Subsequence problem, even for $k = 2$ (see, e.g., [16]).

tably, 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 had to be omitted and are deferred to the full paper.

2 Preliminaries and Previous Work

In this section, we start with a brief introduction to parameterized complexity (more details to be found in [5]) and then continue with sketching some previous work on Closest Substring and Consensus Patterns 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 $O(kn + 1.3^k)$. The corresponding complexity class is called FPT. 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 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 $f(k)n^{O(1)}$, where f might be an arbitrarily fast growing function only depending on k. 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.
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.[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 $\mathbf{N}$ 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 $(x', k') \in L'$.

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

Notably, most reductions from classical complexity turn out *not* to be parameterized ones. Now, the "lowest class of parameterized intractability," W[1], can be defined as the class of languages that reduce 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 complexity theory, and the conjecture that FPT $\neq$ W[1] is very much analogous to the conjecture that P $\neq$ NP. Other problems that are W[1]-*complete* (there are many) include Clique and Independent Set.
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 $f(k)n^{O(1)}$.

2.2 Motivation and Previous Results

Many biological problems with respect to DNA, RNA, or protein sequences can be solved based on consensus word analysis [13, Section 8.6]; Closest Substring and Consensus Patterns are central problems in this context. Applications include locating binding sites and finding conserved regions in unaligned sequences for genetic drug target identification, for designing genetic probes, and for 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 and Consensus Patterns, 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, there is a PTAS for Consensus Patterns [11] as well as for Closest Substring [12], both of which, however, have impractical running times.
Concerning exact (parameterized) algorithms, we only briefly mention that, e.g., Sagot [15] studies motif discovery by solving Closest Substring, Evans and Wareham [6] give FPT algorithms for the same problem, and Blanchette [1] 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 show "exponential behavior with respect to L." To circumvent the computational limitations for larger values of L, many heuristics were proposed, e.g., Pevzner and Sze [14] present algorithms called WINNOWER (wrt. Closest Substring) and SP-STAR (wrt. Consensus Patterns). 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 unlikely to find algorithms with a running time exponential *only* in k.

3 Closest Substring: Unbounded Alphabet

We present a parameterized reduction from Clique to Closest Substring parameterized by the combination of L, d, and k. This shows that Closest Substring is W[1]-hard for the aggregate parameter (L, d, k) in case of unbounded alphabet size.

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 such that G has a clique of size k iff there is a string s of length $L := k + 1$ and 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*. We assume $k > 2$, because $k = 1, 2$ are trivial cases.

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

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

In words, we use a set of *encoding symbols*, i.e., an alphabet symbol for every vertex of the input graph, further we use a unique symbol for every of the $\binom{k}{2}$ produced strings, and we use 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* $S_c = \{c_{1,1}, c_{1,2}, \ldots, c_{1,k}, c_{2,3}, c_{2,4}, \ldots, c_{k-1,k}\}$ and 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 substrings consisting of 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

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

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 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 in order to encode v_r and the jth position is σ_s in order 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

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

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

Proposition 1 *For a graph with a k-clique, the construction in Subsection* 3.1 *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 $d_H(s, s_{i,j}) \le d$.*

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, $1 \le h_1 < h_2 < \ldots < h_k \le n$. 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 $c_{i,j}$, $1 \le i < j \le k$. 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:

$$s_{i,j} \;:= \varphi_{i',1}\,\varphi_{i',2}\cdots\varphi_{i',i-1}\,\sigma_{h_i}\,\varphi_{i',i+1}\cdots\varphi_{i',j-1}\,\sigma_{h_j}\,\varphi_{i',j+1}\cdots\varphi_{i',k}\,\#,$$

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 $1 \le i < j \le k$. □

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 omitted).

Lemma 1 *A solution s has the following properties:*

1. *s contains at least two encoding symbols.*
2. *s contains exactly one "#" symbol, at its last position.*
3. *s does not contain a symbol φ_i, $i = 1,\ldots,\binom{k}{2}$.* □

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, 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, $1 \le i < j \le k$, 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. $\square$

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 each of the single 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. In contrast to there, we are not allowed to 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 complement the set of produced strings by one 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: $0_1 0_2 \ldots 0_{pos-1}\, 1\, 0_{pos+1} \ldots 0_n$, [3] where all $0_1, 0_2, \ldots, 0_n$ stand for "0." In contrast to the reduction from Section 3, now we can 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* $S_c = \{c_{1,1},\, c_{1,2}, \ldots, c_{k-1,k}\}$. 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 $\langle \text{block}() \rangle$ 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, $1 \leq r < s \leq n$, 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 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 $e=(v_r,v_s)$; 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,

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

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:

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

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 $\binom{k}{2}(nk-2k+2)$ 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 $\binom{k}{2}(nk-2k+2)$ 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 $s'''_{i,j}$.

4.2 Correctness of the Reduction

Proposition 3 *For a graph with a k-clique, the construction in Subsection* 4.1 *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 length L substring $s_{i,j}$ with $d_H(s,s_{i,j}) \le d$ and $d_H(s,t) \le d$.*

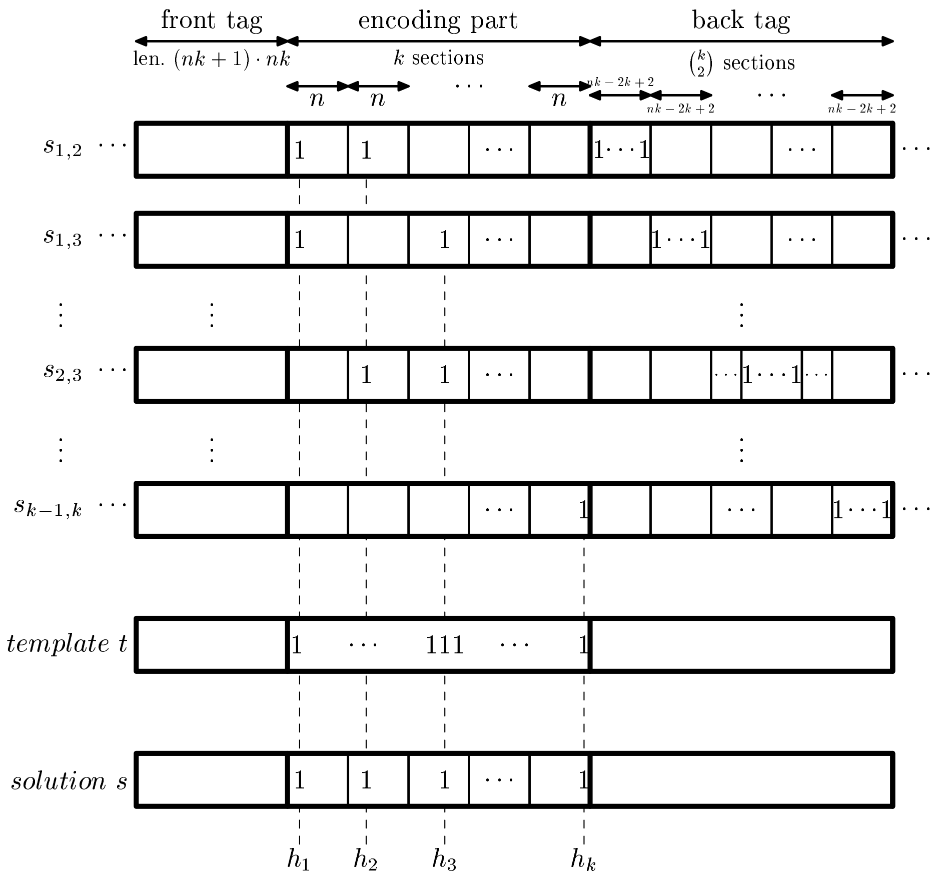

Fig. 1. Illustration of the solution for the Closest Substring instance that is constructed from a graph having a k-clique. We display the template string t, the solution string s, and the matches $s_{i,j}$ in the choice strings. The front tags are equal in all strings and, therefore, are not displayed 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.

$d_H(\cdot,\cdot)$	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	$nk-k$

Fig. 2. Hamming distance of solution s (with front tag s', encoding part s'', and back tag s''') to its matches in the choice strings and the template string.

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 template string, the solution string, and those blocks that are the matches in the choice strings. It follows from the construction that the choice strings have the indicated substrings: 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}. This block is a match for our solution. For this kind of block as well as for the template string, we report in Fig. 2 the distance they have to the solution string, separately for each of their three parts and in total. 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 omitted):

Lemma 2 *A solution s has the following properties:*

1. *s and all its matches in the input instance start with the front tag.*
2. *The encoding part of s contains exactly k "1" symbols.*
3. *Every section of the encoding part of s contains exactly one "1" symbol.* □

Lemma 2 directly implies that the only matches in a choice string $c_{i,j}$ are its blocks and that the solution's back tag 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(1) that s can have as a match only one of the choice string's blocks. By Lemma 2(3), 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. 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 W[1]-Hardness for Consensus Patterns

Our ideas 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 $\sum_{i=1,\ldots,k} d_H(s, s'_i) \leq d$ where s'_i is a length L substring of s_i. Thus, Consensus Patterns aims for minimizing the *sum* of errors.[3] The problem is NP-complete and has a PTAS [11]. 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.

As in Subsection 4.1, we generate a set of $\binom{k}{2}$ choice strings $c_{1,1}, \ldots, c_{k-1,k}$; 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 $0^x(1^x0)^{x+1}0^x$ where $x := (\binom{k}{2}(k-1))nk$. We set parameter L to the block length: $L := x^2 + 4x + 1 + nk$. 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 of nk many "1" symbols. We set $d := (\binom{k}{2} - (k-1))nk$.[4]

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 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. Based on this reduction, we state the following theorem (proof omitted).

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

6 Conclusion

We have proved 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

[3] Here, the most significant parameterization seems to be the one by k, since errors are summed up over all strings and, therefore, d, usually, will not be a small value.

[4] We make sure that a match that is not a block is impossible by letting it, due to the front tags, imply a distance value larger than d. Since d has a higher value here compared to Section 4, we need a more involved front tag.

Common Subsequence [2,3] and Shortest Common Supersequence [9], where parameterized hardness has so far only been established for the case of unbounded alphabet size. 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 no efficient and useful PTAS would be possible [4,5].
Finally, we leave it as an open problem whether Closest Substring and Consensus Patterns, parameterized by k, are in W[1] and, thus, are W[1]-complete — it is easy to see that they are contained in the complexity class $W[P]$.

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. M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64(4):165–171, 1997.
5. R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer. 1999.
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. To appear in *Proc. of 12th ISAAC* (Christchurch, New Zealand), LNCS, December 2001. 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. To appear in *Journal of Computer and System Sciences.*
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. P. A. Pevzner. *Computational Molecular Biology - An Algorithmic Approach.* MIT Press. 2000.
14. 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.
15. 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.
16. D. Sankoff and J. Kruskal (eds.). *Time Warps, String Edits, and Macromolecules.* Addison-Wesley. 1983. Reprinted in 1999 by CSLI Publications.