Finding Consensus Strings With Small Length Difference Between Input and Solution Strings

Markus L. Schmid, Universität Trier

The CLOSEST SUBSTRING PROBLEM is to decide, for given strings $s_1, \ldots, s_k$ of length at most ℓ and numbers m and d, whether there is a length-m string s and length-m substrings s'_i of s_i, such that s has a Hamming distance of at most d from each s'_i. If we instead require the sum of all the Hamming distances between s and each s'_i to be bounded by d, then it is called the CONSENSUS PATTERNS PROBLEM. We contribute to the parameterised complexity analysis of these classical NP-hard string problems by investigating the parameter $(\ell - m)$, i. e., the length difference between input and solution strings. For most combinations of $(\ell - m)$ and one of the classical parameters (m, ℓ, k or d), we obtain fixed-parameter tractability. However, even for constant $(\ell - m)$ and constant alphabet size, both problems remain NP-hard. While this follows from known results with respect to the CLOSEST SUBSTRING, we need a new reduction in the case of the CONSENSUS PATTERNS. As a by-product of this reduction, we obtain an exact exponential-time algorithm for both problems, which is based on an alphabet reduction.

CCS Concepts: •**Theory of computation** → **Problems, reductions and completeness; Fixed parameter tractability;** *W hierarchy;* •**Applied computing** → Computational genomics; Bioinformatics;

Additional Key Words and Phrases: Parameterised complexity, multivariate algorithmics, hard string problems

ACM Reference Format:
Markus L. Schmid, 2016. Finding Consensus Strings With Small Length Difference Between Input and Solution Strings. *ACM Trans. Comput. Theory* 0, 0, Article 0 (0), 19 pages.
DOI:

1. INTRODUCTION

Consensus string problems consist in finding a (preferably long) string that is sufficiently similar to a given set of strings (or to substrings of these given strings). They are among the most classical hard string problems and have many applications, mostly in computationally biology and coding theory (see [Bulteau et al. 2014]). In order to give a mathematically sound definition, we need a measure for the similarity of strings – or rather a distance function – and a classical approach is to use the Hamming distance $\mathsf{d_H}$. In this regard, the central problem considered in this paper is to find, for given strings $s_1, s_2, \ldots, s_k$, a string s of length m that has a Hamming distance of at most d from some length-m substrings $s'_1, s'_2, \ldots, s'_k$ of the input strings.

CLOSEST SUBSTRING

Instance: Strings $s_1, s_2, \ldots, s_k$ over some alphabet Σ with $|s_i| \leq \ell$, $1 \leq i \leq k$, for some $\ell \in \mathbb{N}$, and numbers $m, d \in \mathbb{N}$.
Question: Is there a string s with $|s| = m$ such that, for every i, $1 \leq i \leq k$, s_i has a length-m substring s'_i with $\mathsf{d_H}(s, s'_i) \leq d$?

An extended abstract [Schmid 2015] of this paper was presented at the conference MFCS 2015.
Author's address: Markus L. Schmid, Fachbereich 4 – Abteilung Informatik, Universität Trier, D-54286 Trier, email: mschmid@uni-trier.de.

If we interpret d as an upper bound for the *sum* of all the Hamming distances between s and the substrings s'_i, then we obtain the following variant:

CONSENSUS PATTERNS

Instance: Strings $s_1, s_2, \ldots, s_k$ over some alphabet Σ with $|s_i| \leq \ell$, $1 \leq i \leq k$, for some $\ell \in \mathbb{N}$, and numbers $m, d \in \mathbb{N}$.
Question: Is there a string s with $|s| = m$ such that, for every i, $1 \leq i \leq k$, s_i has a length-m substring s'_i with $\sum_{i=1}^{k} \mathsf{d_H}(s, s'_i) \leq d$?

Both CLOSEST SUBSTRING and CONSENSUS PATTERNS are NP-hard and they have been intensely studied in the multivariate setting (see [Evans et al. 2003], [Fellows et al. 2006], [Frances and Litman 1997], [Marx 2008] and, for a survey, [Bulteau et al. 2014]). The most commonly considered parameters are k, m, d and $|\Sigma|$. The existing results show that CLOSEST SUBSTRING and CONSENSUS PATTERNS are fixed-parameter intractable, even for strong parameterisations. For example, CLOSEST SUBSTRING is W[1]-hard if parameterised by (k, m, d) ([Fellows et al. 2006]) or $(k, d, |\Sigma|)$ ([Marx 2008]). For CONSENSUS PATTERNS, the situation looks slightly better: CONSENSUS PATTERNS parameterised by (k, m, d) or $(k, |\Sigma|)$ is still W[1]-hard ([Fellows et al. 2006]), but it becomes fixed-parameter tractable if parameterised by $(d, |\Sigma|)$ ([Marx 2008]). By simple enumeration, CLOSEST SUBSTRING and CONSENSUS PATTERNS are in FPT with respect to $(m, |\Sigma|)$.

This paper contributes to the multivariate analysis of these consensus string problems by investigating the role of the new parameter $(\ell - m)$, i. e., the length difference between the input strings and the solution string, which can also be seen as an attempt towards Challenge 4 formulated by [Bulteau et al. 2014], which consists in finding new parameters for CLOSEST SUBSTRING that yield fixed-parameter tractability. The relevance of the parameter $(\ell - m)$ is pointed out by the following considerations.

A consensus string problem that is also NP-hard, but which exhibits a better parameterised complexity is CLOSEST STRING, which is defined analogously to CLOSEST SUBSTRING, but with the strong additional restriction that $|s_1| = |s_2| = \ldots = |s_k| = m$. In contrast to CLOSEST SUBSTRING, the problem CLOSEST STRING is fixed-parameter tractable with respect to any of the parameters k, d or m $(= \ell)$. Since CLOSEST STRING can be described as CLOSEST SUBSTRING with the restriction $(\ell - m) = 0$, restricting the parameter $(\ell - m)$ (in the strongest possible way) makes CLOSEST SUBSTRING much easier. In fact, as shall be demonstrated in this work, the fixed-parameter tractability of CLOSEST STRING with respect to k, d and m carries over to CLOSEST SUBSTRING as long as we take $(\ell - m)$ as a second parameter. However, the single parameter $(\ell - m)$ is of not much use, since even if we bound it by 0 and in addition bound the alphabet size by 2, we still have an NP-hard variant of CLOSEST STRING (see [Frances and Litman 1997]).

With respect to CONSENSUS PATTERNS, the parameter $(\ell - m)$ plays a quite different role, since the NP-hardness is not preserved if $(\ell - m) = 0$; in fact, in this case obtaining the closest string is a trivial task. So the question arises, whether bounding $(\ell - m)$ by a constant $c \geq 1$ yields a polynomial-time solvable variant of CONSENSUS PATTERNS or whether it is fixed-parameter tractable with respect to the parameter $(\ell - m)$. In this regard, we can show a strong negative result, namely that CONSENSUS PATTERNS remains NP-hard, even if $(\ell - m) \leq 4$ and $|\Sigma| \leq 4$. Similar to CLOSEST SUBSTRING, some fixed-parameter tractable variants can be obtained by parameterising by $(\ell - m)$ and additional parameters.[1]

[1] A compact presentation of the results of this paper is provided by Tables I to IV.

For proving our main result, i. e., the hardness of CONSENSUS PATTERNS with $(\ell - m) \leq 4$ and $|\Sigma| \leq 4$, we apply an alphabet reduction technique, which also yields an exact exponential-time algorithm for both CLOSEST SUBSTRING and CONSENSUS PATTERNS.

This paper is organised as follows: In Section 2, we define some basic notations and concepts about strings and complexity theory, and we illustrate the consensus string problems in more detail. Then, in Section 3, we present the alphabet reduction technique in terms of an exact exponential-time algorithm for CLOSEST SUBSTRING and CONSENSUS PATTERNS. The following two sections are then devoted to a parameterised complexity analysis of CLOSEST SUBSTRING (Section 4) and CONSENSUS PATTERNS (Section 5) with respect to the parameter $(\ell - m)$. Finally, we conclude this work by Section 6.

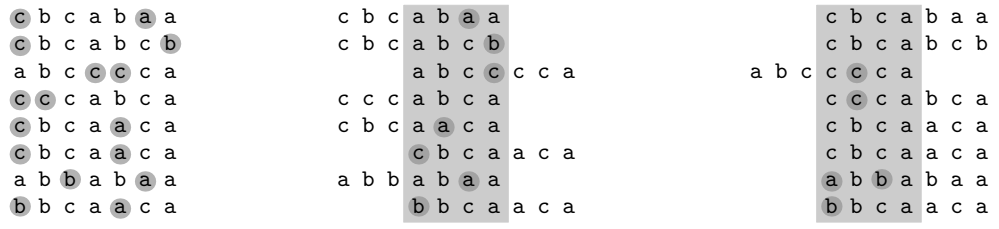
2. PRELIMINARIES

The set of strings over an alphabet Σ is denoted by Σ^*, by $|v|$ we denote the length of a string v, $\mathrm{alph}(v)$ is the smallest alphabet Γ with $v \in \Gamma^*$, a string u is called a *substring* of v, if $v = v'uv''$; if $v' = \varepsilon$ or $v'' = \varepsilon$, then u is a *prefix* or *suffix*, respectively, where ε is the empty string. For a position j, $1 \leq j \leq |v|$, we refer to the symbol at position j of v by the expression $v[j]$ and $v[j..j'] = v[j]v[j+1]\dots v[j']$, $j < j' \leq |v|$. The *Hamming distance* for strings u and v with $|u| = |v|$ is defined by $\mathsf{d_H}(u, v) = |\{j \mid 1 \leq j \leq |u|, u[j] \neq v[j]\}|$.

We assume the reader to be familiar with the basic concepts of (classical) complexity theory. Next, we shall briefly summarise the fundamentals of parameterised complexity (see also [Flum and Grohe 2006]). Decision problems are considered as languages over some alphabet Γ. A *parameterisation* (*of* Γ) is a polynomial-time computable mapping $\kappa : \Gamma^* \to \mathbb{N}$ and a *parameterised problem* is a pair (Q, κ), where Q is a problem (over Γ) and κ is a parameterisation of Γ. We usually define κ implicitly by describing which part of the input is the parameter. A parameterised problem (Q, κ) is *fixed-parameter tractable* if there is an *fpt-algorithm* for it, i. e., an algorithm that solves Q on input x in time $f(\kappa(x)) \times p(|x|)$ for recursive f and polynomial p. The class of fixed-parameter tractable problems is denoted by FPT. Note that if a parameterised problem becomes NP-hard if the parameter is set to a constant, then it is not in FPT unless $\mathsf{P} = \mathsf{NP}$.

For the problems CLOSEST SUBSTRING and CONSENSUS PATTERNS, we consider the parameters k, m, d, $|\Sigma|$, ℓ and $(\ell - m)$, which shall always be denoted in this way. The parameterised versions of the problems are denoted by simply listing the considered parameters in parentheses; if a parameter is bounded by a constant, we explicitly state the constant, e. g., CLOSEST SUBSTRING$(d, (\ell - m))$ is the problem CLOSEST SUBSTRING parameterised by d and $(\ell - m)$, and CONSENSUS PATTERNS$((\ell - m) = c, |\Sigma| = c')$, $c, c' \in \mathbb{N}$, denotes the variant of CONSENSUS PATTERNS, where the parameters $(\ell - m)$ and $|\Sigma|$ are bounded by c and c', respectively.

We now illustrate the concensus string problems defined in Section 1 with some examples. In Figure 1a, a CLOSEST STRING instance with 8 length-7 input strings is represented as an 8×7 matrix. For a Hamming distance bound of $d = 2$, the string $s = \texttt{abcabca}$, shown below the matrix, is a possible solution string, since each input string contains at most two mismatches with s (mismatches are highlighted in grey). If we interpret the same input strings as a CLOSEST SUBSTRING instance with $m = 4$, then we first have to identify a length-4 substring in each of the input strings. Hence, we need a suitable *alignment* of the input strings, i. e., we have to align them in such a way that an 8×4 matrix is obtained (the grey rectangle in Figure 1b). Once such an alignment is fixed, we have to solve CLOSEST STRING for these aligned substrings. If we consider the case $d = 1$, then $\texttt{abca}$ would be a possible solution string (as illustrated



(a) $d = 2$ (b) $m = 4, d = 1$ (c) $m = 4, d = 5$

Fig. 1: CLOSEST STRING, CLOSEST SUBSTRING and CONSENSUS PATTERNS instance.

in Figure 1b). However, the total sum of mismatches is 7, which means that this is not a solution if we interpret the input strings as an instance of CONSENSUS PATTERNS with $d = 5$. In this case, we can choose a different alignment (see Figure 1c) and solution string cbca. It is interesting to note that even though for this alignment and solution string the total number of mismatches is only 5, it is not a solution for CLOSEST SUBSTRING with $d = 1$, since one of the substrings has a Hamming distance of 2. This illustrates the difference between the problems CLOSEST SUBSTRING and CONSENSUS PATTERNS.[2]

The naive approach to solve CLOSEST SUBSTRING or CONSENSUS PATTERNS is to enumerate all possible length-m strings and check whether they are valid solution strings. Obviously, there are an exponential number of strings to check, but checking for a fixed string whether or not it is a valid solution string is not a difficult task. Since we shall use this simple observation at different places in the remainder of this paper, we discuss it here in a bit more detail.

Let $s_1, s_2, \ldots, s_k$ be strings of size at most $\ell \in \mathbb{N}$ over Σ and $m, d \in \mathbb{N}$, and let $w \in \Sigma^*$. For every i, $1 \leq i \leq k$, we compute $\delta_i = \min\{\mathsf{d_H}(w, s[j..j+m-1]) \mid 1 \leq j \leq |s_i| - m + 1\}$, which can be done in time $\mathcal{O}((\sum_{i=1}^{k}(|s_i| - m + 1)m) = \mathcal{O}((\ell - m + 1)mk)$. Then we only have to check whether $\delta_i \leq d$, for every $1 \leq i \leq k$, (in the case of CLOSEST SUBSTRING), or whether $\sum_{i=1}^{k} \delta_i \leq d$ (in the case of CONSENSUS PATTERNS).

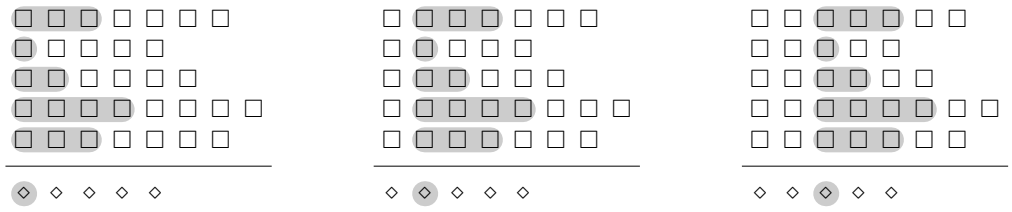
PROPOSITION 2.1. *For a given* CLOSEST SUBSTRING *or* CONSENSUS PATTERNS *instance, we can check in time $\mathcal{O}((\ell - m + 1)mk)$, whether or not a given string is a solution string.*

We conclude this section by introducing some more convenient notation concerning alignments. Position j of the solution string s is said to be *aligned* to position j' of an input string s_i if $s_i[j' - j + 1..j' - j + m]$ is the chosen length-m substring of s_i. Furthermore, we say that s is aligned to s_i at position j' if position 1 of s is aligned to position j' of s_i.

3. ALPHABET REDUCTION AND EXACT EXPONENTIAL-TIME ALGORITHM

In this section we present a technique to transform a CLOSEST SUBSTRING or CONSENSUS PATTERNS instance into an equivalent one by reducing the alphabet over which the input strings are defined. The value of this alphabet reduction is twofold.

[2]In Figure 1, we illustrate the problems CLOSEST STRING, CLOSEST SUBSTRING and CONSENSUS PATTERNS by using the same input strings, in order to point out their differences and similarities. In general, the input strings for CLOSEST SUBSTRING and CONSENSUS PATTERNS can have different lengths.



(a) First symbol (b) Second symbol (c) Third symbol

Fig. 2: Alignment regions for the first, second and third symbol.

Firstly, it is crucial for proving our main result in Section 5.1, where it is used in order to bound the number of symbols that we need in our hardness reduction.

In order to explain its second purpose, we recall that we can solve CLOSEST SUBSTRING or CONSENSUS PATTERNS in time $\mathcal{O}^*(|\Sigma|^m)$ by enumerating all length-m substrings over Σ (see Proposition 2.1).[3] Consequently, reducing the alphabet will improve this running time. Obviously, $|\Sigma| \leq k\ell$; thus, this brute-force algorithm has running-time $\mathcal{O}^*((k\ell)^m)$. In the following, we shall show that the alphabet can always be reduced to a size of at most $k(\ell - m + 1)$, yielding an exact exponential-time algorithm with running-time $\mathcal{O}^*((k(\ell - m + 1))^m)$.

We shall now illustrate the basic idea of the alphabet reduction. To this end, let $s_1, \ldots, s_k \in \Sigma^*$, $m, d \in \mathbb{N}$ be an instance for CLOSEST SUBSTRING or CONSENSUS PATTERNS (in fact, the difference between these problems does not matter in this section). Furthermore, let s be the hypothetical solution string that we are looking for. No matter how we choose our alignment, for every i, $1 \leq i \leq k$, the first symbol of s can only be aligned with positions $1, 2, \ldots, |s_i|-m+1$ of input string s_i. Analogously, the second symbol of s can only be aligned with positions $2, 3, \ldots, |s_i|-m+2$ and so on. This situation is illustrated in Figure 2 for some example instance and the first three symbols of the solution string (since the actual symbols do not matter in these considerations, they are represented by dummy symbols). In this way, every position of the solution string corresponds to an *alignment region* (the grey areas in Figure 2) of the input strings. The mismatches caused by a position of the solution string only depend on the substrings of its alignment region. This suggests that renaming the input strings, such that, for every j, $1 \leq j \leq m$, the structure of the alignment region of position j of the solution string is preserved, yields an equivalent instance.

Before we present this renaming procedure in detail (see Algorithm 1), we sketch it in an intuitive way. Let p the maximum number of symbols that occur in any alignment region and let Σ' be some alphabet of size p. We now injectively rename the first alignment region by symbols from Σ'. Then we consider the second alignment region and note that only its k rightmost symbols are *new*, meaning they are not contained in the first alignment region (thus, not already renamed), whereas the k leftmost symbols of the first alignment region are not contained in the second anymore (see also Figure 2). We then rename the k new symbols in the following way. If a new symbol also occurs somewhere else in the second alignment region, then this new symbol is renamed in the same way as the earlier occurrence. If, on the other hand, the new symbol has no other occurrence in the second alignment region, then we rename it by a symbol from Σ' that is not already used in the part of the second alignment region that has already been renamed. We shall now formally prove that there are always enough symbols in

[3] By $\mathcal{O}^*$ we denote the $\mathcal{O}$-notation that suppresses polynomial factors.

order to carry out this procedure and that it yields an equivalent instance over the alphabet Σ'. After that, we consider an example.

LEMMA 3.1. *Let* $P \in \{\text{CLOSEST SUBSTRING}, \text{CONSENSUS PATTERNS}\}$ *and let* $s_1, s_2, \ldots, s_k \in \Sigma^*$, $m, d \in \mathbb{N}$ *be an instance of* P. *Then there exists an equivalent* P *instance* $t_1, t_2, \ldots, t_k \in \Sigma'^*$, m, d, *with* $|t_i| = |s_i|$, $1 \leq i \leq k$, *and* $|\Sigma'|$ *is of size* $\max\{|\bigcup_{i=1}^{k} \text{alph}(s_i[j..j + (|s_i| - m)])| \mid 1 \leq j \leq m\}$. *The strings* $t_1, t_2, \ldots, t_k$ *can be computed in linear time.*

PROOF. For every i, $1 \leq i \leq k$, we define $\gamma_i = |s_i| - m$. Furthermore, let Σ' be some alphabet with $|\Sigma'| = \max\{|\bigcup_{i=1}^{k} \text{alph}(s_i[j..j + \gamma_i])| \mid 1 \leq j \leq m\}$. In order to understand the following Algorithm 1 (i. e., how it implements the renaming procedure sketched above), it is helpful to keep in mind the purpose of the different data-structures. The stack S contains all symbols from Σ' that are currently free to be used in order to rename symbols from Σ (i. e., they are not already used in the current alignment region), table T on the other hand contains those symbols of Σ that have already been renamed in the current alignment region (along with the symbol from Σ' they are renamed with). The sets A and B store the symbols of the current alignment region of the original strings s_i, whereas the sets C and D store the symbols of the current alignment region of the renamed strings t_i. More precisely, A stores all the symbols on positions that also belong to the previous alignment region, while B stores the new symbols, and C stores all symbols on positions that belong to the next alignment region as well, while D stores only the symbols on the first positions of the alignment regions, i. e., the ones that do not belong to the next alignment region as well.

ALGORITHM 1: Alphabet reduction by renaming

Input : $s_1, s_2, \ldots, s_k$ over Σ
Output: $t_1, t_2, \ldots, t_k$ over Σ'

$t_i = s_i$, $1 \leq i \leq k$;
initialise table $T : \Sigma \to \Sigma' \cup \{\diamond\}$ by $T(a) = \diamond$, for all $a \in \Sigma$;
initialise stack S by $S.\,\text{push}(a)$, for all $a \in \Sigma'$;
construct $A = \bigcup_{i=1}^{k} \text{alph}(s_i[1..\gamma_i])$;
construct $B = \{s_i[\gamma_i + 1] \mid 1 \leq i \leq k\}$;
for *every* i, $1 \leq i \leq k$, j, $1 \leq j \leq \gamma_i + 1$, **do**
    if $T(t_i[j]) = \diamond$ **then**
        $T(t_i[j]) = S.\,\text{pop}$;
    replace $t_i[j]$ by $T(t_i[j])$;
construct $C = \bigcup_{i=1}^{k} \text{alph}(t_i[2..1 + \gamma_i])$;
construct $D = \{t_i[1] \mid 1 \leq i \leq k\}$;
for $j = 2$ **to** m **do**
    for $a \in (D \setminus C)$ **do**
        $S.\,\text{push}(a)$;
    $A = (A \setminus \{s_i[j-1] \mid 1 \leq i \leq k\}) \cup B$;
    $B = \{s_i[j + \gamma_i] \mid 1 \leq i \leq k\}$;
    for $a \in B$ **do**
        if $a \notin A$ **then**
            $T(a) = S.\,\text{pop}$;
        replace all occurrences of a in substrings $t_i[j + \gamma_j]$, $1 \leq i \leq k$, by $T(a)$;
    $D = \{t_i[j] \mid 1 \leq i \leq k\}$;
    $C = (C \setminus D) \cup \{t_i[j + \gamma_i] \mid 1 \leq i \leq k\}$;

As an invariant that is satisfied at the beginning of any iteration of the main loop in Line 12, we formulate the following:

— $A = \bigcup_{i=1}^{k} \mathrm{alph}(s_i[j-1..j-1+\gamma_i-1])$,
— $B = \{s_i[j-1+\gamma_i] \mid 1 \leq i \leq k\}$,
— $C = \bigcup_{i=1}^{k} \mathrm{alph}(t_i[j..j+\gamma_i-1])$,
— $D = \{t_i[j-1] \mid 1 \leq i \leq k\}$ and
— the substrings $t_i[j-1..j-1+\gamma_i]$, $1 \leq i \leq k$, are isomorphic to the substrings $s_i[j-1..j-1+\gamma_i]$, $1 \leq i \leq k$.

It is straightforward to verify that the invariant is satisfied with respect to the properties of the sets A, B, C and D: by Lines 4, 5, 10 and 11, these sets satisfy the claimed properties at the beginning of the first iteration and in Lines 15, 16, 22 and 21, they are updated correctly. It remains to check the last property of the invariant.

At the beginning of the first iteration, the substrings $t_i[1..1+\gamma_i]$, $1 \leq i \leq k$, are isomorphic to the substrings $s_i[1..1+\gamma_i]$, $1 \leq i \leq k$, due to the loop of Line 6, in which the substrings $s_i[1..1+\gamma_i]$, $1 \leq i \leq k$, are renamed in an injective way. Note that this is only possible, since we can assume $|\bigcup_{i=1}^{k} \mathrm{alph}(s_i[1..\gamma_i+1])| \leq |S| = |\Sigma'|$; thus, there are enough symbols in S to perform this renaming. We have to show that the last property of the invariant is maintained by an iteration of the main loop.

In the loop of Line 13, we push back into S all symbols of Σ' that have been used before, but are not present in C, which contains all the renamed symbols of $t_i[j..j+\gamma_i-1]$. Hence, S contains exactly the symbols of $\Sigma' \setminus C$. Then, in Lines 15 and 16, we update the sets A und B, such that $A = \bigcup_{i=1}^{k} \mathrm{alph}(s_i[j..j+\gamma_i-1])$ and $B = \{s_i[j+\gamma_i] \mid 1 \leq i \leq k\}$. Consequently, by the loop in Line 17, in the substrings $s_i[j+\gamma_i]$, $1 \leq i \leq k$, we replace all symbols of $B \cap A$ in the same way as they have been replaced before. All remaining symbols in $B \setminus A$ are replaced by new symbols from S. Let $p = |B \setminus A|$. If $p \leq |S|$, then this leads to substrings $t_i[j..j+\gamma_i]$, $1 \leq i \leq k$, that are isomorphic to the substrings $s_i[j..j+\gamma_i]$. If, on the other hand, $p > |S|$, then $|\Sigma'| < |\bigcup_{i=1}^{k} \mathrm{alph}(t_i[j..j+\gamma_i-1])| + p$. By the inductive hypothesis, the substrings $t_i[j..j+\gamma_i-1]$, $1 \leq i \leq k$, are isomorphic to the substrings $s_i[j..j+\gamma_i-1]$, $1 \leq i \leq k$, which means that $|\bigcup_{i=1}^{k} \mathrm{alph}(t_i[j..j+\gamma_i-1])| + p = |\bigcup_{i=1}^{k} \mathrm{alph}(s_i[j..j+\gamma_i])|$. This is a contradiction, since $|\Sigma'| = \max\{|\bigcup_{i=1}^{k} \mathrm{alph}(s_i[j..j+\gamma_i])| \mid 1 \leq j' \leq m\}$.

The invariant stated above is correct; thus, for every j, $1 \leq j \leq m$, the substrings $t_i[j..j+\gamma_i]$, $1 \leq i \leq k$, are isomorphic to the substrings $s_i[j..j+\gamma_i]$, $1 \leq i \leq k$.

Next, we estimate the running time of Algorithm 1. Let $\alpha = \sum_{i=1}^{k} |s_i|$, i. e., the total size of the input. We first note that B and D can never contain more than k elements. Therefore, we implement the sets B and D as collections of at most k elements and the sets A and C as bit vectors, which means that membership queries, insertions and deletions with respect to A and C can be done in constant time. Initialising T and S requires time $\mathcal{O}(|\Sigma| + |\Sigma'|) = \mathcal{O}(\alpha)$, constructing B and D needs time $\mathcal{O}(k) = \mathcal{O}(\alpha)$ and constructing A and C can be done in time $\mathcal{O}(|\Sigma| + |\Sigma'| + \sum_{i=1}^{k} \gamma_i + 1) = \mathcal{O}(\alpha)$. Furthermore, every iteration of the loop in Line 6 needs constant time; thus, the whole loop requires time $\mathcal{O}(\sum_{i=1}^{k} \gamma_i + 1) = \mathcal{O}(\alpha)$ as well. Hence, the initialisation phase requires time $\mathcal{O}(\alpha)$.

The update operations of the sets A, B, C and D in Lines 15, 16, 22 and 21 can all be done in time $\mathcal{O}(k)$ (since A and C are represented as bit vectors, and B and D have at most k elements). Furthermore, the loop in Line 13 needs time $\mathcal{O}(k)$ as well, and the loop in Line 17 can be implemented in such a way that time $\mathcal{O}(k)$ is sufficient, too.

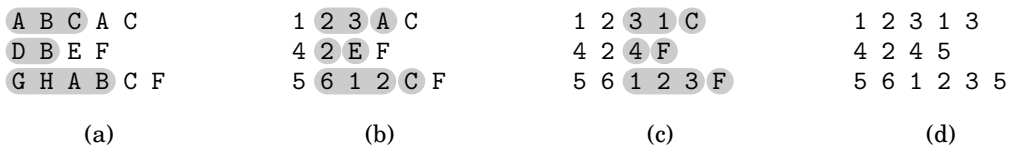


Fig. 3: Renaming of an example instance.

Consequently, the main loop requires time $\mathcal{O}(mk) = \mathcal{O}(\alpha)$. We conclude that the total running time of Algorithm 1 is $\mathcal{O}(\alpha)$.

It remains to prove that the renamed instance $t_1, t_2, \ldots, t_k, m, d$ is equivalent to the original instance $s_1, s_2, \ldots, s_k, m, d$. Let us assume that a length-m string s is aligned with the strings $s_1, s_2, \ldots, s_k$ in some way. For every j, $1 \leq j \leq m$, position j of s must be aligned with a column $x_1 x_2 \ldots x_k$ such that, for every i, $1 \leq i \leq k$, x_i is a symbol from $s_i[j..j + \gamma_i]$, and, without loss of generality, we can also assume that $s[j] \in \{x_1, x_2, \ldots, x_k\}$. Consequently, if we align s in the same way with the strings $t_1, t_2, \ldots, t_k$, then, for every j, $1 \leq j \leq m$, position j of s must be aligned with a column $y_1 y_2 \ldots y_k$ that is isomorphic with $x_1 x_2 \ldots x_k$. Hence, under the considered alignment, the weight of s with respect to strings $s_1, s_2, \ldots, s_k$ is identical to the weight of t with respect to strings $t_1, t_2, \ldots, t_k$, where t has been obtained from s by renaming the symbols according to the renaming that translates the strings s_i to t_i. In the same way, the existence of a string t with weight d with respect to $t_1, t_2, \ldots, t_k$ implies the existence of a string s with weight d with respect to $s_1, s_2, \ldots, s_k$. This directly implies the statement of the lemma. □

As an example for the renaming procedure, we consider the input strings $s_1 = \texttt{ABCAC}$, $s_2 = \texttt{DBEF}$, $s_3 = \texttt{GHABCF}$ over the 8-letter alphabet $\Sigma = \{\texttt{A}, \texttt{B}, \ldots, \texttt{G}, \texttt{H}\}$ and $m = 3$. Figure 3 illustrates how Algorithm 1 renames these strings to strings over the 6-letter alphabet $\Sigma' = \{1, 2, 3, 4, 5, 6\}$. In Figure 3a, the first alignment region is highlighted, which is then consistently renamed as shown in Figure 3b. In Figures 3b and 3c, the second and third alignment regions are highlighted, which consist in a left part that is already renamed and the new symbols to be renamed next. The final result is then shown in Figure 3d.

In order to solve CLOSEST SUBSTRING or CONSENSUS PATTERNS, we can now apply the renaming procedure given by Algorithm 1 and then solve the instance in a brute-force way by enumerating all length m-strings over Σ'. Since $|\Sigma'|$ is bounded by $k(\ell - m + 1)$, we obtain the following result:

THEOREM 3.2. *The problems* CLOSEST SUBSTRING *and* CONSENSUS PATTERNS *can each be solved in time* $\mathcal{O}(k\ell + (k(\ell - m + 1))^m km(\ell - m + 1)) = \mathcal{O}^*((k(\ell - m + 1))^m)$.

As a concluding remark to this section, we note that the above algorithm is also an fpt-algorithm with respect to the parameters k, $(\ell - m)$ and m. However, fixed-parameter tractability with respect to this parameterisation is trivial, since the total input size is bounded in terms of these parameters.

4. CLOSEST SUBSTRING

The parameterised complexity of CLOSEST SUBSTRING is well understood with respect to parameters k, m, d, $|\Sigma|$ and ℓ (see Table I).[4]

[4]In all tables, **p** means that the label of this column is treated as a parameter and an integer entry means that the result holds even if this parameter is set to the given constant; problems that are hard for W[1] are not in FPT (under complexity-theoretical assumptions, see [Flum and Grohe 2006]).

k	m	d	$\|\Sigma\|$	ℓ	Result	Reference
–	–	–	–	**p**	FPT	[Evans et al. 2003]
p	–	–	2	–	W[1]-hard	[Fellows et al. 2006]
p	**p**	**p**	–	–	W[1]-hard	[Fellows et al. 2006]
–	**p**	–	**p**	–	FPT	Trivial
p	–	**p**	2	–	W[1]-hard	[Marx 2008]

Table I: Old results about CLOSEST SUBSTRING.

In the following, we shall take a closer look at the parameter $(\ell - m)$. If we restrict the parameter $(\ell - m)$ in the strongest possible way, i. e., requiring $(\ell - m) = 0$, then the input strings and the solution string have the same length; thus, CLOSEST SUBSTRING collapses to the problem CLOSEST STRING. Unfortunately, CLOSEST STRING is NP-hard even if $|\Sigma| = 2$ (see [Frances and Litman 1997]), which shows the fixed-parameter intractability of CLOSEST SUBSTRING with respect to $(\ell - m)$ and $|\Sigma|$:

PROPOSITION 4.1. CLOSEST SUBSTRING$((\ell - m) = 0, |\Sigma| = 2)$ *is* NP*-hard.*

However, as we shall see next, adding one of k, m or d to the parameter $(\ell - m)$ yields fixed-parameter tractability. For the parameters $(k, (\ell - m))$ and $(m, (\ell - m))$ this can be easily concluded from known results.

THEOREM 4.2. CLOSEST SUBSTRING$(k, (\ell - m))$, CLOSEST SUBSTRING$(m, (\ell - m)) \in$ FPT.

PROOF. Every input string s_i has at most $(\ell - m + 1)$ substrings of length m, so the number of possible alignments of a candidate solution string is at most $(\ell - m + 1)^k$. After an alignment is chosen, the problem is equivalent to solving CLOSEST STRING, which is fixed-parameter tractable if parameterised by k (see [Gramm et al. 2003]). This proves the first statement.

If both m and $(\ell - m)$ are parameters, then also ℓ is a parameter. From CLOSEST SUBSTRING$(\ell) \in$ FPT (see Evans [Evans et al. 2003]), the second statement follows. □

The only case left is the one where $(\ell - m)$ and d are parameters. In comparison to the cases discussed above, an fpt-algorithm for this variant of the problem is more difficult to find. It turns out that an fpt-algorithm for CLOSEST STRING(d) presented in Gramm [Gramm et al. 2003] can be adapted to CLOSEST SUBSTRING$(d, (\ell - m))$.

THEOREM 4.3. CLOSEST SUBSTRING$(d, (\ell - m)) \in$ FPT.

PROOF. Let $s_1, s_2, \ldots, s_k \in \Sigma^*$ and $m, d \in \mathbb{N}$ be a CLOSEST SUBSTRING instance. If a solution string s exists, then it must be possible to construct s by changing at most d symbols in some length-m substring of some s_i. This yields a search tree approach: we start with a length-m substring of s_1 and then we branch into $m|\Sigma|$ new nodes by considering all possibilities of changing a symbol of s into another one. We repeat this procedure d times and for every such constructed string, we check in polynomial-time whether it is a solution string. We shall now improve this procedure such that the branching factor is bounded by $(\ell - m + 1)(d + 1)$, which results in a search tree of size $((\ell - m + 1)(d + 1))^d$.

In a first step, we branch from the root into the at most $(\ell - m + 1)$ substrings of s_1. After that, we branch in every node according to the following rule. Let s' be the string at the current node. We first check whether s' is a solution string in time $\mathcal{O}((\ell - m + 1)mk)$ (see Proposition 2.1). If s' is a solution string, then we can stop. If,

k	m	d	$\lvert\Sigma\rvert$	$(\ell - m)$	Results	Ref.
–	–	–	2	0	NP-hard	[Frances and Litman 1997]
p	–	–	–	**p**	FPT	Thm. 4.2
–	**p**	–	–	**p**	FPT	Thm. 4.2
–	–	**p**	–	**p**	FPT	Thm. 4.3

Table II: New results about CLOSEST SUBSTRING.

on the other hand, s' is not a solution string, then there exists an input string s_i such that all its length-m substrings have too large a distance from s'. Let s'_i be the length-m substring of s_i that is aligned to s (i. e., the assumed solution string). In order to transform s' into s, we have to change $s'[j]$ into $s'_i[j]$ for a position j, $1 \leq j \leq m$, with $s'[j] \neq s'_i[j]$ and $s'_i[j] = s[j]$ (since otherwise this modification cannot lead to s). Since $\mathsf{d_H}(s, s'_i) \leq d$, there are at most d positions j with $s'_i[j] \neq s[j]$; thus, if we choose any $d+1$ positions among all positions j with $s'[j] \neq s'_i[j]$, we will necessarily also select one that satisfies the properties described above (note that there are at least $d+1$ positions with $s'[j] \neq s'_i[j]$, since $\mathsf{d_H}(s', s'_i) > d$). Consequently, for some $A \subseteq \{j \mid 1 \leq j \leq m, s'[j] \neq s'_i[j]\}$, $|A| = d+1$, and every $j \in A$, we construct a new string from s' by changing $s'[j]$ to $s'_i[j]$. This procedure is correct under the assumption from above that s'_i is aligned with s in a solution. Since we have no knowledge of the correct solution alignment, we have to construct $d+1$ new strings for each of the $(\ell - m + 1)$ substrings of s_i, which results in a branching factor of $(\ell - m + 1)(d+1)$.

The total running time of this procedure is $\mathcal{O}((\ell - m + 1)((\ell - m + 1)(d+1))^d k(\ell - m + 1)m) = \mathcal{O}(((\ell - m + 1)(d+1))^{d+1} k(\ell - m + 1)m)$. □

The fixed-parameter tractability results for CLOSEST SUBSTRING established in this section are summarised in Table II.

We now conclude this section by some remarks about Theorem 4.3. The fundamental idea of the algorithm, i. e., changing only $d+1$ symbols in every branching, is the same as for the fpt-algorithm for CLOSEST STRING of [Gramm et al. 2003]. However, to demonstrate that this idea also works for the more general problem CLOSEST SUBSTRING if $(\ell - m)$ is also parameter, and for the sake of self-containment of this paper, it is necessary to present it in a comprehensive way.

In every node, the construction of the successor nodes depends on some s_i, which we are free to choose. Furthermore, the successor nodes can be partitioned into $(\ell - m + 1)$ groups of $(d+1)$ successors that all correspond to the same choice of the length-m substring of s_i. Thus, in the $d+1$ branches of each group, whenever successors are constructed again with respect to s_i, we can always choose the same substring, which results in a branching factor of only $d+1$. Moreover, if we can choose between several s_i's, then we could always select one for which the substring has already been chosen in some predecessor. This heuristic may considerably decrease the size of the search tree.

5. CONSENSUS PATTERNS

Apart from the fact that CONSENSUS PATTERNS$(d, |\Sigma|)$ is fixed-parameter tractable (see [Marx 2008]), CONSENSUS PATTERNS shows a comparatively unfavourable fixed-parameter behaviour in comparison with CLOSEST SUBSTRING (see Table III).

We note that the parameter ℓ is missing from Table III and, to the knowledge of the author, it seems as if this parameter has been neglected in the multivariate analysis of the problem CONSENSUS PATTERNS. Unfortunately, we are not able to answer the most important respective question, i. e., whether or not CONSENSUS PATTERNS$(\ell) \in$

k	m	d	$\|\Sigma\|$	Results	Ref.
p	–	–	2	W[1]-hard	[Fellows et al. 2006]
p	**p**	**p**	–	W[1]-hard	[Fellows et al. 2006]
–	**p**	–	**p**	FPT	Trivial
–	–	**p**	**p**	FPT	[Marx 2008]

Table III: Old results about CONSENSUS PATTERNS.

FPT. Since ℓ is a trivial upper bound for the parameters m and $(\ell - m)$, we state this open question in the following way:

OPEN PROBLEM 1. *Is* CONSENSUS PATTERNS$(\ell, m, (\ell - m))$ *in* FPT*?*

For all other combinations of parameters including ℓ, fixed-parameter tractability can be easily shown:

THEOREM 5.1. CONSENSUS PATTERNS$(|\Sigma|, \ell)$, CONSENSUS PATTERNS(k, ℓ), CONSENSUS PATTERNS$(d, \ell) \in$ FPT.

PROOF. The problem CONSENSUS PATTERNS$(|\Sigma|, m)$ is in FPT, since we can solve it by enumerating all $|\Sigma|^m$ length-m strings over Σ (see also Table III). Since $m \leq \ell$ and $|\Sigma| \leq \ell k$, this directly implies the first two statements.

Obviously, ℓ bounds $(\ell - m)$. Furthermore, CONSENSUS PATTERNS$((\ell - m), d) \in$ FPT (see Theorem 5.2 below), which proves the third statement. □

We shall now turn to the parameter $(\ell - m)$. Unlike as for CLOSEST SUBSTRING, the NP-hardness of CONSENSUS PATTERNS is not preserved if $(\ell - m)$ is bounded by 0. More precisely, if $|s_1| = |s_2| = \ldots = |s_k| = m$, then the length-$m$ string s that minimises $\sum_{i=1}^{k} \mathsf{d_H}(s, s_i)$ is easily constructed by setting $s[j]$, $1 \leq j \leq m$, to one of the symbols that occur the most often among the symbols $s_1[j], s_2[j], \ldots, s_k[j]$.

Consequently, the question arises whether CONSENSUS PATTERNS can still be solved in polynomial-time if $(\ell - m)$ is bounded by larger constants or whether it is fixed-parameter tractable with respect to $(\ell - m)$. Unfortunately, similar to CLOSEST SUBSTRING, CONSENSUS PATTERNS$((\ell - m) = c, |\Sigma| = c')$ is NP-hard, too, for small constants $c, c' \in \mathbb{N}$ (see Theorem5.3). Before we prove this main result of the paper, we consider the other combinations of parameters including $(\ell - m)$.

THEOREM 5.2. CONSENSUS PATTERNS$(k, (\ell - m))$, CONSENSUS PATTERNS$(d, (\ell - m)) \in$ FPT.

PROOF. We can solve CONSENSUS PATTERNS by first choosing length-m substrings $s'_1, s'_2, \ldots, s'_k$ of the input strings and then compute in polynomial-time a length-m string s that minimises $\sum_{i=1}^{k} \mathsf{d_H}(s, s'_i)$ as described above. Since there are at most $(\ell - m + 1)^k$ possibilities of choosing the substrings, the first statement is implied.

In order to prove the second statement, we observe that if $k \leq d$, then we can solve CONSENSUS PATTERNS$(d, (\ell - m))$ by the fpt-algorithm for CONSENSUS PATTERNS$(k, (\ell - m))$. If, on the other hand, $k > d$, then the possible solution string s must be a substring of some input string s_i, since otherwise $\sum_{i=1}^{k} \mathsf{d_H}(s, s'_i) \geq k > d$. Thus, we only have to check the $(\ell - m + 1)k$ length-m substrings of the input strings. □

If CONSENSUS PATTERNS is parameterised by $(\ell - m)$ and m, then we arrive again at the case already mentioned in Open Problem 1. Consequently, there are only two cases left open: the parameter $(\ell - m)$ and the combined parameter $((\ell - m), |\Sigma|)$. We

k	m	d	$(\ell - m)$	$\lvert\Sigma\rvert$	ℓ	Results	Ref.
–	**p**	–	**p**	–	**p**	Open	Open Prob. 1
–	–	–	–	**p**	**p**	FPT	Thm. 5.1
p	–	–	–	–	**p**	FPT	Thm. 5.1
–	–	**p**	–	–	**p**	FPT	Thm. 5.1
–	–	–	4	4	–	NP-hard	Thm. 5.3
p	–	–	**p**	–	–	FPT	Thm. 5.2
–	–	**p**	**p**	–	–	FPT	Thm. 5.2

Table IV: New results about CONSENSUS PATTERNS.

answer the question whether for these cases we have fixed-parameter tractability in the negative, by showing that CONSENSUS PATTERNS remains NP-hard, even if $(\ell - m)$ and $|\Sigma|$ are bounded by small constants.

THEOREM 5.3. CONSENSUS PATTERNS$((\ell - m) = 4, |\Sigma| = 4)$ *is* NP*-complete.*

Before we prove Theorem 5.3 in Section 5.1, we summarise the results (and open questions) of this section in Table IV.

We wish to emphasise here that, in order to prove Theorem 5.3, it is not enough to merely modify existing hardness reductions. This is due to the fact that the existing hardness reductions (see, e. g., [Fellows et al. 2006]) necessarily require the parameter $(\ell - m)$ to be unbounded. More precisely, for a given graph $\mathcal{G} = (V, E)$ and a given number k, the existing reductions transform $\mathcal{G}$ into $\binom{k}{2}$ strings of $|E|$ many blocks each, where every block encodes an edge. The idea is then that a solution string selects an edge of each of those $\binom{k}{2}$ strings in such a way that the selected edges form a k-clique. This selection is done by aligning the solution string with the different blocks of the strings. Consequently, since it must be possible to select any of the edges, it must be possible to align the solution string in at least $|E|$ different ways, which implies that $(\ell - m)$ is necessarily unbounded.

In the next section, we present a different kind of reduction in order to prove Theorem 5.3.

5.1. Proof of Theorem 5.3

We shall prove Theorem 5.3 by a reduction from a variant of the satisfiability problem for Boolean formulas. To this end, we first introduce some notations. The Boolean values are denoted by 0 and 1. We represent a clause of a Boolean formula as a set of literals (i. e., a variable or its negation) and a Boolean formula in conjunctive normal form as a tuple of clauses. For $k \in \mathbb{N}$, a formula in conjunctive normal form is a k-CNF formula if every clause contains at most k literals. An *assignment* for a formula is a mapping $\pi : V \to \{0, 1\}$, where V is the set of variables of the formula. An assignment for a 3-CNF formula is 1*-in-*3 *satisfying*, if it assigns exactly one variable in every clause to 1, and a 3-CNF formula is 1*-in-*3 *satisfiable* if it has a 1-in-3 satisfying assignment. For our reduction, we shall use a variant of the following well-known NP-hard problem (see [Garey and Johnson 1979]).

1-IN-3 3SAT

Instance: A 3-CNF formula C.
Question: Is C 1-in-3 satisfiable?

For our purpose, we need to further restrict this problem, i. e., we require that every clause has exactly 3 literals, that no variable is negated and that every variable occurs

in exactly 3 clauses. By standard modifications of Boolean formulas, we can show that the NP-hardness of 1-IN-3 3SAT is preserved under these conditions.

LEMMA 5.4. 1-IN-3 3SAT *is* NP*-complete for the restriction that every clause has exactly* 3 *literals, no variable is negated and every variable occurs in exactly* 3 *clauses.*

PROOF. Let C_1 be a 3-CNF formula and without loss of generality, we assume that C_1 does not contain clauses with only one literal. We transform C_1 by the following 3 steps:

(1) For every variable x that occurs in negated form, we add a new clause $\{x, x'\}$, where x' is a new variable. Then we replace all negations of x by x'.
(2) We add clauses $\{x_1, x_2, x_3\}$, $\{x_1, x_4, x_5\}$, $\{x_2, x_3, x_5\}$, where every x_i, $1 \leq i \leq 5$, is a new variable. Then we add x_1 to every clause of size 2.
(3) We copy every clause twice, such that there are three identical occurrences of every clause (in particular, this means that every variable x has $3(k_x)$ occurrences, for some $k_x \in \mathbb{N}$). For every variable x with $k = 3\ell > 3$ occurrences, we introduce new variables $\{x_i, y_i, z_i, p_j, q_j, r_j, s_j \mid 1 \leq i \leq k, 1 \leq j \leq \ell\}$ and we add the clauses
— $\{x_1, y_1, z_1\}$, $\{x_1, y_k, z_k\}$, $\{x_i, y_{i-1}, z_{i-1}\}$ and $\{x_i, y_i, z_i\}$, $2 \leq i \leq k$,
— $\{y_{3j-2}, p_j, q_j\}$, $\{y_{3j-1}, p_j, q_j\}$, $\{y_{3j}, p_j, q_j\}$, $1 \leq j \leq \ell$,
— $\{z_{3j-2}, r_j, s_j\}$, $\{z_{3j-1}, r_j, s_j\}$, $\{z_{3j}, r_j, s_j\}$, $1 \leq j \leq \ell$.
Then, for every i, $1 \leq i \leq k$, we substitute the i^{th} occurrence of x by x_i.

Let C_2, C_3 and C_4 be the formulas after Step 1, 2 and 3, respectively.

Claim 1: C_4 is a 3-CNF formula in which every clause has exactly 3 literals, no variable is negated and every variable occurs in exactly 3 clauses.

Proof of Claim 1: In Step 1, we remove all negations and Steps 2 and 3 do not introduce negated variables, so C_4 does not contain negated variables. In Step 2, we only add clauses of size 3 and we turn every clause of size 2 in one of size 3. Furthermore, in Step 3, we only add clauses of size 3. Hence, every clause of C_4 contains exactly 3 variables. It remains to show that the construction of Step 3 produces a formula in which every variable has exactly 3 occurrences. To this end, let x be a variable with $k = 3\ell > 3$ occurrences. We note that each of the new variables y_i, z_i, $1 \leq i \leq k$, p_j, q_j, r_j, s_j, $1 \leq j \leq \ell$, occur in exactly 3 clauses and each of the new variables x_i, $1 \leq i \leq k$, have 2 occurrences in the new clauses and, after replacing all k occurrences of x, one more occurrence in the original part of the formula. Since we apply this construction to every variable with strictly more than 3 occurrences (note that every variable has at least 3 occurrences), all variables in C_4 have exactly 3 occurrences.

(*Claim* 1) □

It remains to prove the correctness of this construction.

Claim 2: C_1 is 1-in-3 satisfiable if and only if C_4 is 1-in-3 satisfiable.

Proof of Claim 2: We prove this claim by showing that, for every i, $1 \leq i \leq 3$, there exists a 1-in-3 satisfying assignment π_{C_i} for C_i if and only if there exists a 1-in-3 satisfying assignment $\pi_{C_{i+1}}$ for C_{i+1}.

Due to the clauses $\{x, x'\}$ in C_2, $\pi_{C_2}(x) \neq \pi_{C_2}(x')$ holds; thus, π_{C_2} is also 1-in-3 satisfying for C_1. Furthermore, π_{C_2} can be obtained from π_{C_1} by extending it such that $\pi_{C_1}(x) \neq \pi_{C_1}(x')$.

Formula C_3 equals C_2 except for the new clauses $\{x_1, x_2, x_3\}$, $\{x_1, x_4, x_5\}$, $\{x_2, x_3, x_5\}$ and occurrences of x_1, which are added to all clauses of size 2. We can obtain π_{C_3} from π_{C_2}, by extending it with $\pi_{C_3}(x_2) = \pi_{C_3}(x_4) = 1$ and $\pi_{C_3}(x_1) = \pi_{C_3}(x_3) = \pi_{C_3}(x_5) = 0$. On the other hand, $\pi_{C_3}(x_1) = 0$ must be satisfied, since $\pi_{C_3}(x_1) = 1$ implies $\pi_{C_3}(x_2) = \pi_{C_3}(x_3) = \pi_{C_3}(x_4) = \pi_{C_3}(x_5) = 0$, which leads to the contradiction that $\{x_2, x_3, x_5\}$ is

not 1-in-3 satisfied. Hence, in every clause containing x_1, one of the other two variables must be assigned to 1 by π_{C_3}, so π_{C_3} is also 1-in-3 satisfying for C_2.

We can extend π_{C_3} to π_{C_4} in the following way. We set $\pi_{C_4}(x_i) = \pi_{C_3}(x)$, $1 \leq i \leq k$. If $\pi_{C_3}(x) = 1$, then $\pi_{C_4}(y_i) = \pi_{C_4}(z_i) = 0$, $1 \leq i \leq k$, $\pi_{C_4}(p_j) = \pi_{C_4}(r_j) = 0$ and $\pi_{C_4}(q_j) = \pi_{C_4}(s_j) = 1$, $1 \leq j \leq \ell$, and if $\pi_{C_3}(x) = 0$, then $\pi_{C_4}(y_i) = 1$, $\pi_{C_4}(z_i) = 0$, $1 \leq i \leq k$, $\pi_{C_4}(p_j) = \pi_{C_4}(q_j) = \pi_{C_4}(r_j) = 0$ and $\pi_{C_4}(s_j) = 1$, $1 \leq j \leq \ell$. On the other hand, $\pi_{C_4}(x_1) = \pi_{C_4}(x_2) = \ldots = \pi_{C_4}(x_k)$ must be satisfied, which can be seen as follows. If $\pi_{C_4}(x_i) = 1$, then, due to $\{x_i, y_i, z_i\}$, both $\pi_{C_4}(y_i) = \pi_{C_4}(z_i) = 0$. This means that, due to $\{x_{i+1}, y_i, z_i\}$, $\pi_{C_4}(x_{i+1}) = 1$. If, on the other hand, $\pi_{C_4}(x_i) = 0$, then either $\pi_{C_4}(y_i) = 1$ or $\pi_{C_4}(z_i) = 1$, which means that $\pi_{C_4}(x_{i+1}) = 0$. By induction, it follows that all x_i are assigned to the same value, which means that π_{C_3} can be obtained from π_{C_4} by setting $\pi_{C_3}(x) = \pi_{C_4}(x_1)$. (*Claim* 2) □

This concludes the proof of the lemma. □

Remark 5.5. It is a well-known fact that 1-IN-3 3SAT remains NP-hard, even if no variable is negated and every clause has size 3 (see [Garey and Johnson 1979]). However, to the knowledge of the author, this further restricted case from above is not discussed in the literature.

Tailored to the reduction to be introduced next, we define our clauses to be sets, whereas the whole formula is a tuple, such that clauses can be repeated. However, this does not constitute a loss of generality. If we consider clauses that can contain a variable more than once, then (since we are looking for 1-in-3 satisfying assignments) we could remove these variables by assigning them to 0. Furthermore, the hardness of 1-IN-3 3SAT with no repeated clauses trivially carries over to the case, where clauses can be repeated.

For the sake of convenience, in the following, we represent an assignment for a Boolean formula by the set of variables that are assigned to 1. In this regard, there exists a 1-in-3 satisfying assignment for a 3-CNF formula $C = (c_1, c_2, \ldots, c_n)$ over a set of variables V and without negations if and only if there exists a subset $D \subseteq V$, such that, for every i, $1 \leq i \leq n$, $|D \cap c_i| = 1$.

We now define a reduction from this restricted version of 1-IN-3 3SAT to CONSENSUS PATTERNS. To this end, let $C = (c_1, c_2, \ldots, c_n)$ be an instance of 1-IN-3 3SAT, where no variable is negated, every variable occurs in exactly three clauses and the set of variables is $V = \{v_1, v_2, \ldots, v_{\widehat{n}}\}$. In order to define the clauses in a more convenient way, we use mappings $\wp_r : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, \widehat{n}\}$, $1 \leq r \leq 3$, that map an $i \in \{1, 2, \ldots, n\}$ to the index of the r^{th} variable (with respect to some arbitrary order) of clause c_i, i. e., for every i, $1 \leq i \leq n$, $c_i = \{v_{\wp_1(i)}, v_{\wp_2(i)}, v_{\wp_3(i)}\}$.

The formula C shall now be transformed into strings over $\Sigma = V \cup \{\star\}$ as follows.[5] For every i, $1 \leq i \leq n$, c_i is transformed into

$$s_i = \star^4\, t_{i,1}\, t_{i,2}\, \ldots\, t_{i,n}\, \star^4, \text{with}$$
$$t_{i,j} = \alpha_{i,j,1} \star \alpha_{i,j,2} \star \alpha_{i,j,3} \star, \text{for every } j, 1 \leq j \leq n,$$

where, for every r, $1 \leq r \leq 3$, $\alpha_{i,j,r} = v_{\wp_r(i)}$ if $v_{\wp_r(i)} \in c_j$ and $\alpha_{i,j,r} = \star$ otherwise. Furthermore, for every r, $1 \leq r \leq 3$, we construct a string

$$q_r = \star^4\, v_{\wp_r(1)}\, \star^5\, v_{\wp_r(2)}\, \star^5 \ldots v_{\wp_r(n)}\, \star^5\ .$$

[5]The size of Σ obviously depends on $|V|$ and therefore is not constant; we shall later show how our construction can be modified in such a way that an alphabet of size 4 is sufficient.

Moreover, $m = 6n + 4$ and $d = 3n^2 + 5n$. Note that $\ell = 6n + 8$; thus, $(\ell - m) = 4$. The CONSENSUS PATTERNS instance, denoted by I_C, consists now of the strings $s_1, s_2, \ldots, s_n$ and $n + 1$ copies of each of the strings q_r, $1 \leq r \leq 3$.

Let us explain this reduction in an intuitive way and illustrate it with an example. For every i, $1 \leq i \leq n$, the string s_i contains the substrings $t_{i,j}$, which each is a list of exactly the variables from $c_i \cap c_j$. However, how the elements $c_i \cap c_j$ are represented in $t_{i,j}$ is crucial. If, for example, exactly the first and third variable of c_i is also in c_j, i. e., $c_i \cap c_j = \{v_{\wp_1(i)}, v_{\wp_3(i)}\}$, then $t_{i,j} = v_{\wp_1(i)} \star\star\star v_{\wp_3(i)} \star$. Thus, for every i, $1 \leq i \leq n$, $t_{i,i}$ represents the complete clause c_i, i. e., $t_{i,i} = v_{\wp_1(i)} \star v_{\wp_2(i)} \star v_{\wp_3(i)} \star$, whereas every $t_{i,j}$, $1 \leq j \leq n$, $i \neq j$, equals $t_{i,i}$ after all variables have been erased (i. e., replaced by $\star$) that are not in $c_i \cap c_j$. In order to illustrate this, let $c_{i_1} = \{v_1, v_3, v_7\}$, $c_{i_2} = \{v_3, v_4, v_1\}$, $c_{i_3} = \{v_2, v_1, v_8\}$, $c_{i_4} = \{v_7, v_8, v_9\}$ and $c_{i_5} = \{v_{10}, v_7, v_3\}$ be example clauses, for some $1 \leq i_j \leq n$. Then $s_{i_1} = \star^4 t_{i_1,1} t_{i_1,2} \ldots t_{i_1,n} \star^4$ with

$$\begin{aligned}
t_{i_1,i_1} &= v_1 \star v_3 \star v_7 \star ,\\
t_{i_1,i_2} &= v_1 \star v_3 \star \star \star ,\\
t_{i_1,i_3} &= v_1 \star \star \star \star \star ,\\
t_{i_1,i_4} &= \star \star \star \star v_7 \star ,\\
t_{i_1,i_5} &= \star \star v_3 \star v_7 \star ,
\end{aligned}$$

and $t_{i_1,j} = \star^6$, for every j, $1 \leq j \leq n$, $j \notin \{i_2, i_3, i_4, i_5\}$.

Before moving on, we first introduce more convenient notations in order to facilitate the following technical statements. Let s be a solution string for some CONSENSUS PATTERNS instance. For every j, $1 \leq j \leq m$, the *weight of position j (of s)* is the number of mismatches between $s[j]$ and the aligned symbols in the input strings. Hence, s is a solution string if its *total weight*, i. e., the sum of the weights of all positions, is at most d. If s has minimal weight among all possible solution strings, then it is an *optimal solution string*. We note that the strings q_r of I_C have a length of m; thus, the alignment of a candidate solution string s only concerns the strings s_i. For every j, $1 \leq j \leq n$, we define the position $\delta_j = 5 + 6(j - 1)$, i. e., the δ_j are the positions of the strings q_r that contain a symbol from V and the positions of the strings s_i, where a substring $t_{i,j}$ starts. The following observation shall simplify our further argumentation.

OBSERVATION 1. *For every i, $1 \leq i \leq n$, s_i has exactly 9 occurrences of symbols from V. More precisely, if $c_i = \{x_1, x_2, x_3\}$, then, for every r, $1 \leq r \leq 3$, s_i has exactly 3 occurrences of x_r at positions $\delta_{j_1} + 2(r - 1)$, $\delta_{j_2} + 2(r - 1)$ and $\delta_{j_3} + 2(r - 1)$, for some j_1, j_2, j_3, $1 \leq j_1, j_2, j_3 \leq n$.*

Next, we will show that an optimal solution string for I_C necessarily has a certain structure, from which we can later conclude the existence of 1-in-3 satisfying assignment for C.

LEMMA 5.6. *Let s be an optimal solution string for the instance I_C. Then $s = \star^4 v_{p_1} \star^5 v_{p_2} \star^5 \ldots v_{p_n} \star^5$ with $v_{p_i} \in c_i$, $1 \leq i \leq n$, and s is aligned to s_i at position $1, 3$ or 5, $1 \leq i \leq n$.*

PROOF. We assume that s and the way it is aligned results in a total weight of $d' \leq d$. By a sequence of separate claims, we show that s satisfies the structure claimed in the satement of the lemma. We start with the symbols at positions δ_i, $1 \leq i \leq n$.

Claim 1: $s[\delta_i] \in c_i$, for every i, $1 \leq i \leq n$.

Proof of Claim 1: We assume to the contrary that, for some i, $1 \leq i \leq n$, $s[\delta_i] \notin c_i$. The symbol $s[\delta_i]$ is aligned to a symbol from c_i in every q_r and to a symbol from $c_i \cup \{\star\}$ in every s_j (the latter is due to the fact that, since $(\ell - m) = 4$, s must be aligned to

s_j at one of the positions $1, 2, \ldots, 5$, and therefore position δ_i of s must be aligned with one of the positions of $t_{j,i}$). Thus, position δ_i of s contributes at least $3n + 3$ to the total weight, if $s[\delta_i] = \star$ and at least $4n + 3$, if $s[\delta_i] \in V \setminus c_i$. If we change $s[\delta_i]$ to $v_{\varphi_1(i)} \in c_i$, then it matches the aligned symbol in all $n + 1$ copies of q_1. Hence, in the worst case, it can only cause mismatches with respect to the remaining $(2n + 2) + n = 3n + 2$ strings, so $s[\delta_i]$ contributes at most $3n + 2$ to the total weight, which is a contradiction to the optimality of s. (*Claim* 1) □

Next, we prove that all remaining positions of s must carry the symbol $\star$.

Claim 2: $s[j] = \star$, for every j, $1 \leq j \leq m$, with $j \notin \{\delta_1, \delta_2, \ldots, \delta_n\}$.
Proof of Claim 2: If, for some j, $1 \leq j \leq m$, with $j \notin \{\delta_1, \delta_2, \ldots, \delta_n\}$, $s[j] \neq \star$, then position j contributes at least $3n + 3$ to the total weight, since it constitutes a mismatch with the aligned symbol in all strings q_r. If we change $s[j]$ to $\star$, then position j contributes a weight of at most n, since it matches with respect to all strings q_r and can only have mismatches with respect to the n strings s_i. This is a a contradiction to the optimality of s. (*Claim* 2) □

Finally, it only remains to show that the solution string s can only be aligned at certain positions, which can be concluded from its structure, established by the previous claims.

Claim 3: For every i, $1 \leq i \leq n$, s is aligned to s_i at position $1, 3$ or 5.
Proof of Claim 3: Due to the fact that $(\ell - m) = 4$, the only possible positions where s can be aligned to some s_i are $1, 2, \ldots, 5$. We now assume that s is aligned to some s_i at position 2 or 4 and we recall that s contains exactly n occurrences of symbols from V and the rest are occurrences of $\star$, while s_i contains exactly 9 occurrences of symbols from V and the rest are occurrences of $\star$ (see Observation 1). Hence, the maximum number of mismatches between s and s_i is $n + 9$. If s is aligned to s_i at position 2 or 4, then, for every j, $1 \leq j \leq n$, position δ_j is aligned to the 2^{nd} or 4^{th} position of $t_{i,j}$, which is $\star$. Moreover, every occurrence of a symbol from V in s_i is aligned to some symbol of s (this is due to the fact that s_i starts and ends with $\star^4$) and since all symbols from V in s are already aligned to symbol $\star$ of s_i, all symbols from V in s_i must be aligned with occurrences of $\star$ of s. This yields the maximum number of $n + 9$ mismatches between s and s_i. If, on the other hand, s is aligned at a position $1, 3$ or 5, then, since $t_{i,i} = v_{\varphi_1(i)} \star v_{\varphi_2(i)} \star v_{\varphi_3(i)} \star$, position $s[\delta_i]$ is necessarily aligned to a symbol from V in s_i, which implies that the total number of mismatches must be strictly less than $n + 9$ (in fact, we can also conclude that there are at least two more positions δ_j that must be aligned with a symbol from V in s_i). This is a contradiction to the optimality of s. (*Claim* 3) □

This conludes the proof of the lemma. □

We are now ready to show that the existence of a solution string (which, by the previous lemma, implies the existence of a solution string of the form $\star^4 v_{p_1} \star^5 v_{p_2} \star^5 \ldots v_{p_n} \star^5$, with $v_{p_i} \in c_i$, $1 \leq i \leq n$), implies the existence of a 1-in-3 satisfying assignment of C. The idea is to collect all symbols v_{p_j}, $1 \leq j \leq n$, and then show that if v_{p_j} (which is in c_j) is also in c_i, then v_{p_i} is necessarily aligned to an occurrence of v_{p_j} in s_j, which allows us to argue that, in order to not exceed the bound d on the Hamming distance, $v_{p_i} = v_{p_j}$ must hold. From this property, it follows that the set $\{v_{p_1}, v_{p_2}, \ldots, v_{p_n}\}$ is a 1-in-3 satisfying assignment.

LEMMA 5.7. *If there exists a solution string, then C is 1-in-3 satisfiable.*

PROOF. We assume that s is an optimal solution string. With Lemma 5.6, we can conclude that $s = \star^4 v_{p_1} \star^5 v_{p_2} \star^5 \ldots v_{p_n} \star^5$ with $v_{p_i} \in c_i$, $1 \leq i \leq n$, and s is aligned to s_i

at position $1, 3$ or 5, $1 \leq i \leq n$. In the following two claims, we count the mismatches between s and the input strings.

Claim 1: The number of mismatches between s and all copies of the strings q_r, $1 \leq r \leq 3$, is $2n^2 + 2n$.

Proof of Claim 1: For every i, $1 \leq i \leq n$, the symbol at position δ_i of s matches with the aligned symbol of exactly one of the 3 strings q_r, $1 \leq r \leq 3$, and therefore causes $2(n+1)$ mismatches. All other positions $j \notin \{\delta_1, \delta_2, \ldots, \delta_n\}$ of s are matching the aligned positions of the strings q_r, $1 \leq r \leq 3$. Thus, the number of mismatches between s and all copies of the strings q_r, $1 \leq r \leq 3$, is $2(n+1)n = 2n^2 + 2n$. (*Claim* 1) □

Claim 2: For every i, $1 \leq i \leq n$, the number of mismatches between s and s_i is $n + 3$.

Proof of Claim 2: Due to Claim 1, there are $2n^2 + 2n$ mismatches between s and all copies of the strings q_r, $1 \leq r \leq 3$. Thus, since s is a solution string, there are at most $d - (2n^2 + 2n) = n^2 + 3n = n(n+3)$ mismatches between s and all the strings s_i, $1 \leq i \leq n$. For every i, $1 \leq i \leq n$, s is aligned to s_i at a position $r \in \{1, 3, 5\}$, which implies that every δ_j, $1 \leq j \leq n$, is aligned to the r^{th} position of $t_{i,j}$. However, only for j, $1 \leq j \leq n$, with $v_{\wp_{\frac{r+1}{2}}(i)} \in c_j$ (of which there are only 3), the r^{th} position of $t_{i,j}$ carries a symbol different from $\star$. Hence, $(n-3)$ positions δ_j of s are aligned to a symbol $\star$ in s_i and, consequently, at most 3 of the 9 occurrences of symbols from V in s_i (see Observation 1) can be aligned to a symbol different from $\star$ in s. This results in at least $(n-3)+6 = n+3$ mismatches between s and s_i. Together with the fact that the total number of mismatches between s and all the strings s_i, $1 \leq i \leq n$, is at most $n(n+3)$, this implies that, for every i, $1 \leq i \leq n$, the number of mismatches between s and s_i is $n+3$. (*Claim* 2) □

We define $D = \{v_{p_1}, v_{p_2}, \ldots, v_{p_n}\}$. In order to conclude that D is a 1-in-3 satisfying assignment, we have to show that, for every i, $1 \leq i \leq n$, $|c_i \cap D| = 1$, which is established by the following claim.

Claim 3: Let $i \in \{1, 2, \ldots, n\}$, $r \in \{1, 3, 5\}$, $u = \frac{r+1}{2}$ and let j_1, j_2, j_3, $1 \leq j_1 < j_2 < j_3 \leq n$, be such that $v_{\wp_u(i)} \in c_{j_1} \cap c_{j_2} \cap c_{j_3}$. If s is aligned to s_i at position r, then $v_{p_{j_1}} = v_{p_{j_2}} = v_{p_{j_3}} = v_{\wp_u(i)}$.

Proof of Claim 3: We first assume that s is aligned with s_i at position $r = 1$ (note that this means that $u = 1$). Every v_{p_j}, $1 \leq j \leq n$, is aligned with position δ_j of s_i, i. e., with the first symbol of $t_{i,j}$. By the structure of s_i, only the 3 positions δ_{j_1}, δ_{j_2}, δ_{j_3} of s_i carry a symbol from V, namely $v_{\wp_1(i)}$, whereas all other $n-3$ positions $\delta_{j'}$ with $j' \notin \{j_1, j_2, j_3\}$ of s_i carry the symbol $\star$. Hence, there are $n-3$ mismatches resulting from these positions $\delta_{j'}$. Moreover, all other 6 occurrences of symbols from V in s_i (i. e., the ones not corresponding to a position δ_j) constitute mismatches with an occurrence of $\star$ in s. Consequently, there are already $n+3$ mismatches between s and s_i, which, according to Claim 2, are all the mismatches between s and s_i. In particular, this means that the positions δ_{j_1}, δ_{j_2} and δ_{j_3} of s_i constitute matches, which implies that $s[\delta_{j_1}] = s[\delta_{j_2}] = s[\delta_{j_3}] = v_{\wp_1(i)}$, which means that $v_{p_{j_1}} = v_{p_{j_2}} = v_{p_{j_3}} = v_{\wp_u(i)}$. The cases $r \in \{3, 5\}$ can be handled analogously; the only difference is that positions δ_j of s are aligned with positions $\delta_j + (r-1)$ of s_i. (*Claim* 3) □

From this claim, it follows that if $v_{p_i} \in c_j$ for some i, j, $1 \leq i, j \leq n$, $i \neq j$, then $v_{p_i} = v_{p_j}$. Hence, if, for some j, $1 \leq j \leq n$, $|c_j \cap D| \geq 2$, then there exists an i, $1 \leq i \leq n$, $j \neq i$, with $v_{p_i} \in c_j$ and $v_{p_j} \neq v_{p_i}$, which is a contradiction. Consequently, $|D \cap c_i| = 1$, for every i, $1 \leq i \leq n$; thus, D is a 1-in-3 satisfying assignment. □

It remains to prove the other direction, i. e., that a 1-in-3 satisfying assignment for C can be translated into a solution string for I_C.

LEMMA 5.8. *If C is 1-in-3 satisfiable, then there exists a solution string.*

PROOF. Let $D = \{v_{p_1}, v_{p_2}, \ldots, v_{p_n}\}$ be a 1-in-3 satisfying assignment with $v_{p_j} \in c_j$, $1 \leq j \leq n$. We shall show that $s = \star^4 v_{p_1} \star^5 v_{p_2} \star^5 \ldots v_{p_n} \star^5$, which is of length m, is a solution string. First, we measure the weight caused by the strings q_r. All occurrences of $\star$ in s match the aligned symbols in the strings q_r and therefore the only mismatches can occur at positions δ_i. However, for every i, $1 \leq i \leq n$, $s[\delta_i] = v_{p_i}$ matches the aligned symbol in exactly 1 of the strings q_r, $1 \leq r \leq 3$, because $v_{p_i} \in c_i$. Consequently, all copies of the strings q_r contribute a weight of $2n(n+1)$.

For every i, $1 \leq i \leq n$, we align s with s_i at position $2r-1$, where r is such that $v_{p_i} = v_{\wp_r(i)}$, i. e., we align v_{p_i} of s with the matching symbol of $t_{i,i}$ in s_i. By the structure of s_i, this implies that every symbol at a position δ_j with $v_{p_i} \in c_j$ matches the aligned symbol in s_i and all other $n-3$ positions δ_j with $v_{p_i} \notin c_j$ are aligned with $\star$ and are therefore mismatches. All the remaining 6 occurrences of symbols from V in s_i must also be aligned with symbols of s (due to the fact that s_i starts and ends with $\star^4$). Moreover, they all are aligned with occurrences of $\star$ and therefore constitute mismatches as well. Since all remaining positions are matches, we conclude that the strings s_i contribute a weight of $n(n-3+6)$, which leads to a total weight of $2n(n+1) + n(n-3+6) = 3n^2 + 5n = d$. □

In order to complete the proof of Theorem 5.3, it remains to show how the alphabet size can be bounded by 4. To this end, we first slightly modify the reduction by adding occurrences of the string $\star^4$ to every s_i, $1 \leq i \leq n$, between the substrings $t_{i,j}$ and $t_{i,j+1}$, $1 \leq j \leq n-1$, and to every q_r, $1 \leq r \leq 3$, between the substrings $v_{\wp_r(j)} \star^5$ and $v_{\wp_r(j+1)} \star^5$, i. e.,

$$s_i = \star^4 t_{i,1} \star^4 t_{i,2} \star^4 \ldots t_{i,n} \star^4, 1 \leq i \leq n\,,$$
$$q_r = \star^4 v_{\wp_r(1)} \star^5 \star^4 v_{\wp_r(2)} \star^5 \star^4 \ldots v_{\wp_r(n)} \star^5, 1 \leq r \leq 3\,.$$

Consequently, also the positions δ_j slightly change, i. e., $\delta_j = 5 + 10(j-1)$, $1 \leq j \leq n$. Furthermore, we set $m = 6n + 4 + 4(n-1) = 10n$ and $d = 3n^2 + 5n$. We note that $\ell = 6n + 8 + 4(n-1) = 10n + 4$ and therefore $(\ell - m) = 4$. This reduction is still correct (in fact, the proofs apply in the same way).

For every i, $1 \leq i \leq n$, r, $1 \leq r \leq 3$, and j, $1 \leq j \leq m$, we define

$$\Gamma_{i,j} = \mathrm{alph}(s_i[j..j + (|s_i| - m)]), \text{ and}$$
$$\Delta_{r,j} = \mathrm{alph}(q_r[j..j + (|q_r| - m)])\,.$$

This means that, using the terminology of Section 3, for every j, $1 \leq j \leq m$, $(\bigcup_{i=1}^{n} \Gamma_{i,j}) \cup (\bigcup_{r=1}^{3} \Delta_{r,j})$ is the alphabet of the alignment region of position j of the solution string. By Lemma 3.1, the modified instance from above can be transformed into an equivalent one over an alphabet of size $\sigma = \max\{|(\bigcup_{i=1}^{n} \Gamma_{i,j}) \cup (\bigcup_{r=1}^{3} \Delta_{r,j})| \mid 1 \leq j \leq m\}$. Hence, we have to show that $\sigma = 4$.

We first note that, for every j, $|s_i[j..j + (|s_i| - m)]| = 5$, $1 \leq i \leq n$, and $|q_r[j..j + (|q_r| - m)]| = 1$, $1 \leq r \leq 3$. Consequently, for every j, $1 \leq j \leq m$, and r, $1 \leq r \leq 3$, $(\bigcup_{r=1}^{3} \Delta_{r,j}) = \{\star\}$ if $j \notin \{\delta_1, \delta_2, \ldots, \delta_n\}$ and $(\bigcup_{r=1}^{3} \Delta_{r,j}) = c_i$ if $j = \delta_i$. Furthermore, for every i, $1 \leq i \leq n-1$, and j, $\delta_i - 4 \leq j \leq \delta_{i+1} - 5$, $(\bigcup_{i=1}^{n} \Gamma_{i,j}) \subseteq c_i \cup \{\star\}$ and, for every j, $\delta_n - 4 \leq j \leq m$, $(\bigcup_{i=1}^{n} \Gamma_{i,j}) \subseteq c_n \cup \{\star\}$. Hence, for every j, $1 \leq j \leq m$, $|(\bigcup_{i=1}^{n} \Gamma_{i,j})| \leq 4$ and, since $(\bigcup_{r=1}^{3} \Delta_{r,j}) \subseteq (\bigcup_{i=1}^{n} \Gamma_{i,j})$, also $|(\bigcup_{i=1}^{n} \Gamma_{i,j}) \cup (\bigcup_{r=1}^{3} \Delta_{r,j})| \leq 4$. We conclude that $\sigma \leq 4$, which implies (with Lemma 3.1) that the modified instance can be transformed into an equivalent one over an alphabet of size 4. This finally concludes the proof of Theorem 5.3.

6. CONCLUSIONS

We investigated the role of the parameter $(\ell - m)$ for the consensus string problems CLOSEST SUBSTRING and CONSENSUS PATTERNS. In a sense, this parameter measures the difference between a CLOSEST SUBSTRING instance and an instance of the easier problem CLOSEST STRING (or between a CLOSEST SUBSTRING instance and an instance of the trivial variant of CONSENSUS PATTERNS, where every input string has length m). While parameterising by $(\ell - m)$ and at least one additional parameter typically allows fpt-algorithms, the parameter $(\ell - m)$ alone is too weak in order to obtain fixed-parameter tractability.

In this regard, our main negative result is the NP-hardness of CONSENSUS PATTERNS$((\ell - m) = 4, |\Sigma| = 4)$ (and, thus, the fixed-parameter intractability of CONSENSUS PATTERNS$((\ell - m), |\Sigma|)$, assuming P $\neq$ NP). This result improves the one that is reported in the extended abstract [Schmid 2015] of this paper, in which we could only show the NP-hardness of CONSENSUS PATTERNS$((\ell - m) = 6, |\Sigma| = 5)$. Although this seems like a small difference, the stronger result presented here is a significant improvement, since consensus string problems are mainly motivated by applications on DNA-strings, which have an alphabet of size 4. However, this stronger result, too, leaves a gap with respect to smaller constant bounds. More precisely, it is open whether CONSENSUS PATTERNS$((\ell - m) = c, |\Sigma| = c')$, can be solved in polynomial-time for constants c and c', where c or c' is strictly smaller than 4.

Acknowledgements

The author wishes to thank Katrin Casel for valuable comments and discussions on the improvement of Theorem 5.3.

REFERENCES

L. Bulteau, F. Hüffner, C. Komusiewicz, and R. Niedermeier. 2014. Multivariate Algorithmics for NP-Hard String Problems. *Bulletin of the EATCS* 114 (2014), 31–73.

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

M. R. Fellows, J. Gramm, and R. Niedermeier. 2006. On The Parameterized Intractability Of Motif Search Problems. *Combinatorica* 26 (2006), 141–167.

J. Flum and M. Grohe. 2006. *Parameterized Complexity Theory*. Springer-Verlag New York, Inc.

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

M. R. Garey and D. S. Johnson. 1979. *Computers And Intractability*. W. H. Freeman and Company.

J. Gramm, R. Niedermeier, and P. Rossmanith. 2003. Fixed-Parameter Algorithms for CLOSEST STRING and Related Problems. *Algorithmica* 37 (2003), 25–42.

D. Marx. 2008. Closest Substring Problems with Small Distances. *SIAM J. Comput.* 38 (2008), 1382–1410.

M. L. Schmid. 2015. Finding Consensus Strings with Small Length Difference Between Input and Solution Strings. In *Proc. 40th International Symposium on Mathematical Foundations of Computer Science (Part II), MFCS 2015 (LNCS)*, Vol. 9235. 542–554.