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

Markus L. Schmid

Trier University, Fachbereich IV – Abteilung Informatikwissenschaften,
D-54286 Trier, Germany, MSchmid@uni-trier.de

Abstract. The parameterised complexity of the Closest Substring Problem and the Consensus Patterns Problem with respect to the parameter $(\ell - m)$ is investigated, where ℓ is the maximum length of the input strings and m is the length of the solution string. We present an exact exponential time algorithm for both problems, which is based on an alphabet reduction. Furthermore, it is shown that for most combinations of $(\ell - m)$ and one of the classical parameters (m, ℓ, number of input strings k, distance d), we obtain fixed-parameter tractability, but even for constant $(\ell - m)$ and constant alphabet size, both problems are NP-hard.

Keywords: Parameterised complexity, hard string problems

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 [1]). 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 least d from some length-m substrings $s'_1, s'_2, \ldots, s'_k$ of the input strings.

Closest Substring (CloseSubstr)

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 substring s'_i with $\mathsf{d_H}(s, s'_i) \leq d$?

If we require $\sum_{i=1}^{k} \mathsf{d_H}(s, s'_i) \leq d$ instead of $\mathsf{d_H}(s, s'_i) \leq d$, $1 \leq i \leq k$, then the problem is called Consensus Patterns (ConsPat).

Both CloseSubstr and ConsPat are NP-hard and they have been intensely studied in the multivariate setting (see [2, 3, 5, 8] and, for a survey, [1]). The most commonly considered parameters are k, m, d and $|\Sigma|$. The existing results show that CloseSubstr and ConsPat are fixed-parameter intractable, even for

highly parameterised cases. For example, CloseSubstr is W[1]-hard if parameterised by (k, m, d) [3] or $(k, d, |\Sigma|)$ [8]. For ConsPat, the situation looks slightly better: ConsPat parameterised by (k, m, d) or $(k, |\Sigma|)$ is still W[1]-hard [3], but it becomes fixed-parameter tractable if parameterised by $(d, |\Sigma|)$ [8]. By simple enumeration, CloseSubstr and ConsPat are in FPT with respect to $(m, |\Sigma|)$.

In contrast to that, an NP-hard consensus string problem that exhibits a better parameterised complexity (see [6]), is the Closest String Problem (CloseStr), which is similar to CloseSubstr (the "computationally harder sister problem of CloseStr", according to [1]), with the strong additional restriction that $|s_1| = |s_2| = \ldots = |s_k| = m$. Analogously, we can also define CloseStr as CloseSubstr with the restriction $(\ell - m) = 0$. The NP-hardness of CloseStr implies that bounding $(\ell - m)$ by a constant does not yield polynomial time solvability of CloseSubstr; thus, the question arises whether some of the positive fixed-parameter tractability results for CloseStr carry over to CloseSubstr and ConsPat if $(\ell - m)$ is considered a parameter. This paper is devoted to an investigation of this parameter $(\ell - m)$, which can also be seen as an attempt towards Challenge 4 formulated in [1], which consists in finding new parameters of CloseSubstr that yield fixed-parameter tractability.

Our Contribution[1] We first present an exact exponential time algorithm for CloseSubstr and ConsPat, based on an alphabet reduction, that runs in time $\mathcal{O}^*((k(\ell - m + 1))^m)$.[2]

For CloseSubstr, parameter $(\ell - m)$ alone cannot lead to fixed-parameter tractability, since even for $(\ell - m) = 0$ and $|\Sigma| = 2$ (i. e., CloseStr with binary alphabet) the problem remains NP-hard (see [5]). However, it is comparatively easy to show that in fact the tractable cases of CloseStr carry over to CloseSubstr if we additionally take $(\ell - m)$ as parameter (the parameter $((\ell - m), d)$ requires a bit more work, for which we adapt an fpt-algorithm for CloseStr parameterised by d presented in [6]).

For ConsPat, the situation is more complicated. Firstly, setting $(\ell - m) = 0$ makes the problem easily polynomial time solvable. In order to answer the question about its fixed-parameter tractability with respect to $((\ell - m), |\Sigma|)$ in the negative, we conduct a new reduction for which the alphabet reduction of the exact exponential algorithm mentioned above shall play an important role. For parameters $((\ell - m), k)$ and $((\ell - m), d)$ fixed-parameter tractability follows from simple enumeration algorithms, but the case of parameter $((\ell - m), m)$ is open. Obviously, the combined parameter $((\ell - m), m)$ is equivalent to parameter ℓ, which, to the knowledge of the author, has been neglected in the multivariate analysis of ConsPat. In this regard, we can at least note that parameter ℓ leads to fixed-parameter tractability if any of k, d or $|\Sigma|$ is also treated as a parameter.

Due to space constraints, not all results are formally proven.

Basic Definitions 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 Γ with $v \in \Gamma^*$,

[1] A compact presentation of all results is provided by Tables 1 and 2.

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

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 [4]). 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 $\mathcal{O}(f(\kappa(x)) \times p(|x|))$ 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 CloseSubstr and ConsPat, 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., CloseSubstr$(d, (\ell - m))$ is the problem CloseSubstr parameterised by d and $(\ell - m)$), and ConsPat$((\ell - m) = c, |\Sigma| = c')$, $c, c' \in \mathbb{N}$, denotes the variant of ConsPat, where the parameters $(\ell - m)$ and $|\Sigma|$ are bounded by c and c', respectively.

We conclude this section by introducing some more convenient terminology. Let $s_1, s_2, \dots, s_k$ and m, d be an instance of CloseSubstr or ConsPat, and let s be a fixed candidate for a solution string. We say that s is *aligned* with s_i at position j, $1 \leq j \leq |s_i| - m + 1$, in order to denote that s is compared to the substring $s_i[j..j+m-1]$. Once we have fixed such an alignment, every single position j of s is aligned with (or *corresponds* to) a position of every input string, which can either be a *match* or a *mismatch*. In the case of ConsPat, it is also convenient to interpret position j to be aligned with the *column* $x_1 x_2 \dots x_k$, where x_i is the aligned symbol of s_i. Every position j has then between 0 and k mismatches with respect to its corresponding column.

2 Alphabet Reduction and Exact Exponential Algorithm

The problems CloseSubstr and ConsPat can both be solved by enumerating all length-m strings over the input alphabet Σ and check for each such string whether or not is a solution string, which can be done in time $\mathcal{O}(km(\ell - m + 1))$. This yields an algorithm with running time $\mathcal{O}^*(|\Sigma|^m)$ or, since $|\Sigma| \leq k\ell$, $\mathcal{O}^*((k\ell)^m)$. In this section, we improve this naive algorithm by an alphabet reduction, such that a running time of $\mathcal{O}^*((k(\ell - m + 1))^m)$ is achieved.

In order to illustrate the basic idea of this alphabet reduction, let $s_1, \ldots, s_k \in \Sigma^*$, $m, d \in \mathbb{N}$ be a CloseSubstr or ConsPat instance. For every i, $1 \leq i \leq k$, we define $\gamma_i = |s_i| - m$. Since a solution string s can be aligned with s_i only at positions $1, 2, \ldots, \gamma_i + 1$, any position j of s can only be aligned with one of the positions $j, j+1, \ldots, \gamma_i + j$ of s_i. Thus, regardless of the actual alignment, the mismatches caused by position j only depend on the substrings $s_i[j..j+\gamma_i]$. This suggests that renaming the strings s_i, such that, for every j, $1 \leq j \leq m$, the structure of the substrings $s_i[j..j+\gamma_i]$, $1 \leq i \leq k$, is preserved, yields an equivalent instance. We sketch such a renaming procedure.

For every j, $1 \leq j \leq m$, let $\Gamma_j = \bigcup_{i=1}^{k} \mathrm{alph}(s_i[j..j+\gamma_i-1])$ and $\Delta_j = \{s_i[j+\gamma_i] \mid 1 \leq i \leq k\}$. Let Σ' be some alphabet with $|\Sigma'| = \max\{|\Gamma_j \cup \Delta_j| \mid 1 \leq j \leq m\}$. For all values $j = 1, 2, \ldots, m$, we injectively rename (i. e., different symbols are replaced by different symbols) the substrings $s_i[j..j+\gamma_i]$, $1 \leq i \leq k$, with symbols from Σ'. However, since these substrings overlap, in every step j, $j \geq 2$, the substrings $s_i[j..j+\gamma_i-1]$ are already renamed and therefore, except for the first step, we only have to rename the length-1 substrings $s_i[j+\gamma_i]$, which can be done as follows. For all i, $1 \leq i \leq k$, if $s_i[j+\gamma_i] \in \Gamma_j$, then we rename $s_i[j+\gamma_i]$ as has been done before in the substrings $s_i[j..j+\gamma_i-1]$. All the remaining *new* symbols $\Delta_j \setminus \Gamma_j$ are injectively renamed by some symbols from $\Sigma' \setminus \bigcup_{i=1}^{k} \mathrm{alph}(s_i[j..j+\gamma_i-1])$.

For the strings $s_1 = abcac$, $s_2 = dbef$, $s_3 = ghabcf$ and $m = 3$, we have $\gamma_1 = 2$, $\gamma_2 = 1$, $\gamma_3 = 3$ and $\max\{|\bigcup_{i=1}^{3} \mathrm{alph}(s_i[j..j+\gamma_i])| \mid 1 \leq j \leq 3\} = 6$. For $\Sigma' = \{A, B, \ldots, F\}$, the renaming described above proceeds as follows:

$$\begin{array}{llllll}
s_1 = & a\ b\ c\ a\ c & & A\ B\ C\ a\ c & & A\ B\ C\ A\ c & & A\ B\ C\ A\ C & = t_1 \\
s_2 = & d\ b\ e\ f & \Rightarrow & D\ B\ e\ f & \Rightarrow & D\ B\ D\ f & \Rightarrow & D\ B\ D\ E & = t_2 \\
s_3 = & g\ h\ a\ b\ c\ f & & E\ F\ A\ B\ c\ f & & E\ F\ A\ B\ C\ f & & E\ F\ A\ B\ C\ E & = t_3
\end{array}$$

This reduces the alphabet by 2 symbols and it can be easily verified that, for every j, $1 \leq j \leq 3$, the substrings $t_1[j..j+\gamma_1], t_2[j..j+\gamma_2]$ and $t_3[j..j+\gamma_3]$ are isomorphic to the substrings $s_1[j..j+\gamma_1], s_2[j..j+\gamma_2]$ and $s_3[j..j+\gamma_3]$. From this property, we can also conclude that the instances s_1, s_2, s_3 and t_1, t_2, t_3 are equivalent; thus, we obtain the following result.

Lemma 1. *Let $P \in \{\text{CloseSubstr}, \text{ConsPat}\}$ 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 $\max\{|\bigcup_{i=1}^{k} \mathrm{alph}(s_i[j..j+(|s_i|-m)])| \mid 1 \leq j \leq m\}$. The strings $t_1, t_2, \ldots, t_k$ can be computed in time $\mathcal{O}(|\Sigma| + k\ell)$.*

An instance of CloseSubstr or ConsPat can now be solved by first reducing the alphabet to Σ' by the renaming procedure and then solve the instance by checking for every length m-string over Σ' whether it is a solution string. Since $|\Sigma'|$ is bounded by $k(\ell - m + 1)$, we obtain the following result:

Theorem 1. *The problems CloseSubstr and ConsPat can each be solved in time $\mathcal{O}(|\Sigma| + k\ell + (k(\ell-m+1))^m km(\ell-m+1)) = \mathcal{O}^*((k(\ell-m+1))^m)$.*

This alphabet reduction can also be interpreted as a *kernelisation* with respect to the parameters $k, (\ell - m), m$. However, with respect to these parameters, fixed-parameter tractability can be shown more directly. Therefore, Lemma 1 has no application in proving fixed-parameter tractability results, but it shall be an important tool later on in Section 4 for proving a hardness result.

3 Closest Substring

The parameterised complexity of CloseSubstr is well understood with respect to parameters k, m, d, $|\Sigma|$ and ℓ (see left side of Table 1).[3]

k	m	d	$\lvert\Sigma\rvert$	ℓ	Result	Reference
–	–	–	–	**p**	FPT	[2]
p	–	–	2	–	W[1]-hard	[3]
p	**p**	**p**	–	–	W[1]-hard	[3]
–	**p**	–	**p**	–	FPT	Trivial
p	–	**p**	2	–	W[1]-hard	[8]

k	m	d	$\lvert\Sigma\rvert$	$(\ell - m)$	Results	Ref.
–	–	–	2	0	NP-hard	Prop. 1
p	–	–	–	**p**	FPT	Thm. 2
–	**p**	–	–	**p**	FPT	Thm. 2
–	–	**p**	–	**p**	FPT	Thm. 3

Table 1. Old and new results about CloseSubstr.

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, CloseSubstr collapses to the problem CloseStr. Unfortunately, CloseStr is NP-hard even if $|\Sigma| = 2$ (see [5]), which shows the fixed-parameter intractability of CloseSubstr with respect to $(\ell - m)$ and $|\Sigma|$:

Proposition 1. CloseSubstr$((\ell - m) = 0, |\Sigma| = 2)$ *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 2. CloseSubstr$(k, (\ell - m))$, CloseSubstr$(m, (\ell - m)) \in$ 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 CloseStr, which is fixed-parameter tractable if parameterised by k (see [6]). This proves the first statement.

If both m and $(\ell - m)$ are parameters, then also ℓ is a parameter. From CloseSubstr$(\ell) \in$ FPT (see [2]), the second statement follows. □

[3] 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 [4]).

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 CloseStr(d) presented in [6] can be adapted to the problem CloseSubstr$(d, (\ell - m))$.

Theorem 3. CloseSubstr$(d, (\ell - m)) \in \mathsf{FPT}$.

Proof. Let $s_1, s_2, \ldots, s_k \in \Sigma^*$ and $m, d \in \mathbb{N}$ be a CloseSubstr 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 $\mathcal{O}(k(\ell - m + 1)m)$. If s' is a solution string, then we can stop. If, 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 $\mathcal{O}((\ell - m + 1)((\ell - m + 1)(d + 1))^d k(\ell - m + 1)m) = \mathcal{O}(((\ell - m + 1)(d + 1))^{d+1} k(\ell - m + 1)m)$. □

We conclude this section by some remarks about Theorem 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 CloseStr of [6]. However, to demonstrate that if $(\ell - m)$ is also parameter, then this idea works for the more general problem CloseSubstr, too, 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 can considerably decrease the size of the search tree.

4 Consensus Patterns

Apart from ConsPat$(d, |\Sigma|) \in$ FPT (see [8]), ConsPat shows a comparatively unfavourable fixed-parameter behaviour as CloseSubstr (left side of Table 2).

k	m	d	$\lvert\Sigma\rvert$	Results	Ref.
p	–	–	2	W[1]-hard	[3]
p	**p**	**p**	–	W[1]-hard	[3]
–	**p**	–	**p**	FPT	Trivial
–	–	**p**	**p**	FPT	[8]

k	m	d	$(\ell-m)$	$\lvert\Sigma\rvert$	ℓ	Results	Ref.
–	**p**	–	**p**	–	**p**	Open	Open Prob. 4
–	–	–	–	**p**	**p**	FPT	Thm. 5
p	–	–	–	–	**p**	FPT	Thm. 5
–	–	**p**	–	–	**p**	FPT	Thm. 5
–	–	–	6	5	–	NP-hard	Thm. 7
p	–	–	**p**	–	–	FPT	Thm. 6
–	–	**p**	**p**	–	–	FPT	Thm. 6

Table 2. Old and new results about ConsPat.

We note that the parameter ℓ is missing from the left side of Table 2 and, to the knowledge of the author, it seems as though this parameter has been neglected in the multivariate analysis of the problem ConsPat. Unfortunately, we are not able to answer the most important respective question, i. e., whether or not ConsPat$(\ell) \in$ FPT. Since ℓ is a trivial upper bound for the parameters m and $(\ell - m)$, we state this open question in the following form:

Open Problem 4 *Is* ConsPat$(\ell, m, (\ell - m))$ *in* FPT*?*

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

Theorem 5. ConsPat$(|\Sigma|, \ell)$, ConsPat(k, ℓ), ConsPat$(d, \ell) \in$ FPT.

Proof. The problem ConsPat$(|\Sigma|, m)$ is in FPT (see left side of Table 2). Since $m \leq \ell$ and $|\Sigma| \leq \ell k$, this directly implies the first two statements.

Obviously, ℓ bounds $(\ell - m)$. Furthermore, ConsPat$((\ell - m), d) \in$ FPT (see Theorem 6 below), which proves the third statement. □

4.1 The Parameter $(\ell - m)$

We shall now turn to the parameter $(\ell - m)$. Unlike as for CloseSubstr, the NP-hardness of ConsPat 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 $s_1[j], s_2[j], \ldots, s_k[j]$.

Nevertheless, similar to CloseSubstr, ConsPat$((\ell - m) = c, |\Sigma| = c')$ is NP-hard, too, for small constants $c, c' \in \mathbb{N}$ (see Theorem7). Before we prove this main result of the paper, we consider the other combinations of parameters.

Theorem 6. ConsPat$(k, (\ell - m))$, ConsPat$(d, (\ell - m)) \in$ FPT.

Proof. We can solve ConsPat 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 ConsPat$(d, (\ell - m))$ by the fpt-algorithm for ConsPat$(k, (\ell - m))$. 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 ConsPat is parameterised by $(\ell - m)$ and m, then we arrive again at the problem already mentioned in Open Problem 4. Consequently, there are only two cases left open: the parameter $(\ell - m)$ and the combined parameter $((\ell - m), |\Sigma|)$. We answer the question whether for these cases we have fixed-parameter tractability in the negative, by showing that ConsPat remains NP-hard, even if $(\ell - m)$ and $|\Sigma|$ are small constants.

Theorem 7. ConsPat$((\ell - m) = 6, |\Sigma| = 5)$ *is* NP*-hard.*

The existing reductions proving hardness results for ConsPat (see [3]) construct $\binom{k}{2}$ strings representing the same graph $\mathcal{G} = (V, E)$ by listing its edges. A solution string then selects an edge from each string such that the selected edges form a k-clique. Thus, it must be possible to align the solution string with $|E|$ different substrings. This means that $\ell - m$ necessarily depends on $|E|$ and therefore this general idea of reduction is unsuitable for our case.

We choose a different approach, but we also use a graph problem, for which we first need the following definitions. Let $\mathcal{G} = (V, E)$ be a graph with $V = \{v_1, v_2, \ldots, v_n\}$. A vertex s is the *neighbour* of a vertex t if $\{t, s\} \in E$ and $N_{\mathcal{G}}[t] = \{s \mid \{t, s\} \in E\} \cup \{t\}$ is called the *closed neighbourhood* of t (or simply *neighbourhood*, for short). If, for some $k \in \mathbb{N}$, every vertex of $\mathcal{G}$ has exactly k neighbours, then $\mathcal{G}$ is *k-regular*. A *perfect code* for $\mathcal{G}$ is a subset $C \subseteq V$ with $|N_{\mathcal{G}}[t] \cap C| = 1$, $t \in V$. Next, we define the problem to decide whether or not a given 3-regular graph has a perfect code, which is NP-hard (see [7]):

3-Regular Perfect Code (3RPerCode)
Instance: A 3-regular graph $\mathcal{G}$.
Question: Does $\mathcal{G}$ contain a perfect code?

We now define a reduction from 3RPerCode to ConsPat. To this end, let

$\mathcal{G} = (V, E)$ be a 3-regular graph with $V = \{v_1, v_2, \ldots, v_n\}$ and, for every i, $1 \leq i \leq n$, N_i is the neighbourhood of v_i. In order to define the neighbourhoods in a more convenient way, we use mappings $\wp_r : \{1, 2 \ldots, n\} \to \{1, 2 \ldots, n\}$, $1 \leq r \leq 4$, that map an $i \in \{1, 2 \ldots, n\}$ to the index of the r^{th} vertex (with respect to some arbitrary order) of neighbourhood N_i, i. e., for every i, $1 \leq i \leq n$, $N_i = \{v_{\wp_1(i)}, v_{\wp_2(i)}, v_{\wp_3(i)}, v_{\wp_4(i)}\}$.

We now transform $\mathcal{G}$ into strings over $\Sigma = V \cup \{\star\}$. The size of $|\Sigma|$ 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 5 is sufficient. For every i, $1 \leq i \leq n$, N_i is transformed into $s_i = \star^6 t_{i,1} t_{i,2} \ldots t_{i,n} \star^6$ with $t_{i,i} = v_{\wp_1(i)} \star v_{\wp_2(i)} \star v_{\wp_3(i)} \star v_{\wp_4(i)} \star$ and, for every j, $1 \leq j \leq n$, $i \neq j$, $t_{i,j} = \alpha_1 \star \alpha_2 \star \alpha_3 \star \alpha_4 \star$, where, for every r, $1 \leq r \leq 4$, $\alpha_r = v_{\wp_r(i)}$ if $v_{\wp_r(i)} \in N_j$ and $\alpha_r = \star$ otherwise. Furthermore, for every r, $1 \leq r \leq 4$, we construct a string $q_r = \star^6 v_{\wp_r(1)} \star^7 v_{\wp_r(2)} \star^7 \ldots v_{\wp_r(n)} \star^7$. Moreover, $m = 8n + 6$ and $d = 4n^2 + 11n$. Note that $\ell = 8n + 12$; thus, $(\ell - m) = 6$. The ConsPat instance consists now of the strings $s_1, s_2, \ldots, s_n$ and $n + 1$ copies of each of the strings q_r, $1 \leq r \leq 4$.

Let us explain this reduction in an intuitive way and illustrate it with an example. For every i, $1 \leq i \leq n$, N_i is completely represented in the string s_i by $t_{i,i} = v_{\wp_1(i)} \star v_{\wp_2(i)} \star v_{\wp_3(i)} \star v_{\wp_4(i)} \star$. Furthermore, every $t_{i,j}$ with $i \neq j$ contains exactly the vertices from $N_i \cap N_j$, but they are listed according to N_i, i. e., every $t_{i,j}$ corresponds to the list $v_{\wp_1(i)} \star v_{\wp_2(i)} \star v_{\wp_3(i)} \star v_{\wp_4(i)} \star$ in which all elements not in N_j have been erased (i. e., replaced by $\star$). In the solution strings, there will be n special positions, each of which is aligned with position $1, 3, 5$ or 7 of $t_{i,j}$; thus, selecting one of the 4 possible vertices of $t_{i,j}$ (or $\star$ if no vertex is present). If $v_{\wp_r(i)}$ is selected from $t_{i,i}$, then also the r^{th} vertex in every $t_{i,j}$, $i \neq j$, must be selected. Due to the order in which the vertices are listed, this is either the exact same vertex $v_{\wp_r(i)}$ in case that $v_{\wp_r(i)} \in N_j$ or $\star$ otherwise.

The 3-regularity of $\mathcal{G}$ allows us to bound $|t_{i,j}|$; thus, bounding $(\ell - m)$ as well. Moreover, it implies that in every s_i there are exactly 16 occurrences of vertices, which we exploit in the definition of the distance bound d. We illustrate these definitions with an example: let $N_1 = (v_1, v_4, v_5, v_8)$, $N_4 = (v_5, v_9, v_4, v_1)$, $N_5 = (v_1, v_5, v_{10}, v_4)$, $N_8 = (v_1, v_8, v_{11}, v_{15})$. Then $s_1 = \star^6 t_{1,1} t_{1,2} \ldots t_{1,n} \star^6$ with

$$t_{1,1} = v_1 \star v_4 \star v_5 \star v_8 \star, \quad t_{1,4} = v_1 \star v_4 \star v_5 \star\star\star, \quad t_{1,5} = v_1 \star v_4 \star v_5 \star\star\star,$$
$$t_{1,8} = v_1 \star\star\star\star\star v_8 \star, \quad t_{1,9} = \star\star v_4 \star\star\star\star\star, \quad t_{1,10} = \star\star\star\star v_5 \star\star\star,$$
$$t_{1,11} = \star\star\star\star\star\star v_8 \star, \quad t_{1,15} = \star\star\star\star\star\star v_8 \star\,.$$

In addition to these strings s_i, we use $n + 1$ copies of the length-m strings q_r,

$$\begin{aligned}
q_1 &= \star^6\ v_1\ \star^7\ v_{\wp_1(2)}\ \star^7\ \ldots\ v_{\wp_1(n)}\ \star^7,\\
q_2 &= \star^6\ v_4\ \star^7\ v_{\wp_2(2)}\ \star^7\ \ldots\ v_{\wp_2(n)}\ \star^7,\\
q_3 &= \star^6\ v_5\ \star^7\ v_{\wp_3(2)}\ \star^7\ \ldots\ v_{\wp_3(n)}\ \star^7,\\
q_4 &= \star^6\ v_8\ \star^7\ v_{\wp_4(2)}\ \star^7\ \ldots\ v_{\wp_4(n)}\ \star^7,
\end{aligned}$$

which contain the neighbourhoods in form of columns separated by symbols $\star$ and serve the purpose of enforcing a certain structure of the solution string.

Before moving on, we first introduce more convenient notations in order to facilitate the following technical statements. Since the strings q_r have a length of m, the alignment of a candidate solution string s only concerns the strings s_i. 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 corresponding 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. For every i, $1 \leq i \leq n$, we define the position $\delta_i = 8(i-1)+7$, i. e., the δ_i are the positions of the strings q_r that contain a symbol from V and the positions of the strings s_j, where a substring $t_{j,i}$ starts.

We have to show that $\mathcal{G}$ has a perfect code if and only if there is a solution string for the ConsPat instance. The next lemma proves the *only if* direction, which is the easier one.

Lemma 2. *If $\mathcal{G}$ has a perfect code, then there exists a solution string.*

How a solution string translates into a perfect code is more difficult to show. To this end, we first observe that the string s with the lowest possible weight necessarily adheres to a certain structure, which, in conjunction with the fact that it has a weight of at most d, allows then to extract a perfect code for $\mathcal{G}$.

Lemma 3. *If there exists a solution string, then $\mathcal{G}$ has a perfect code.*

Proof. Without loss of generality, we assume that s and the way it is aligned results in a total weight of $d' \leq d$ that is minimal among all possible total weights. By a sequence of separate claims, we show that s has a certain structure.

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

Proof of Claim: We assume to the contrary that, for some i, $1 \leq i \leq n$, $s[\delta_i] \notin N_i$. The symbol $s[\delta_i]$ corresponds to a symbol from N_i in every q_r and to a symbol from $N_i \cup \{\star\}$ in every s_j (the latter is due to the fact that position δ_i of s must be aligned with a position of $t_{j,i}$). Thus, position δ_i of s contributes at least $4n+4$ to the total weight, if $s[\delta_i] = \star$ and at least $5n+4$, if $s[\delta_i] \in V \setminus N_i$. If we change $s[\delta_i]$ to $v_{\wp_1(i)} \in N_i$, then it matches the corresponding symbol in all $n+1$ copies of q_1. Hence, it contributes at most $(4n+4)-(n+1)+n = 4n+3$ to the total weight, if $s[\delta_i] = \star$ and at most $(5n+4)-(n+1) = 4n+3$, if $s[\delta_i] \in V \setminus N_i$. This is a contradiction to the minimality of d'. □

Claim: $s[j] = \star$, for every j, $1 \leq j \leq m$, with $j \notin \{\delta_1, \delta_2, \ldots, \delta_n\}$.

Proof of Claim: 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 $4n+4$ to the total weight, since it constitutes a mismatch with the corresponding 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 contradiction to the minimality of d'. □

Claim: For every i, $1 \leq i \leq n$, s is aligned with s_i at position $1, 3, 5$ or 7.

Proof of Claim: The only possible positions where s can be aligned with some s_i are $1, 2, \ldots, 7$. If s is aligned with some s_i at position $2, 4$ or 6, then, for every

j, $1 \leq j \leq n$, position δ_j corresponds to the $2^{\text{nd}}, 4^{\text{th}}$ or 6^{th} position of $t_{i,j}$, which is $\star$. Since $s[\delta_j] \in N_j$, $1 \leq j \leq n$, all n occurrences of symbols from V in s cause mismatches. Moreover, all 16 occurrences of symbols from V in s_i correspond to occurrences of $\star$ in s. This yields $n+16$ mismatches between s and s_i. If s is aligned at a position $1, 3, 5$ or 7, then it is still possible that all the n symbols at positions δ_j, $1 \leq j \leq n$, cause mismatches, but since 4 of these positions correspond to occurrences of symbols from V in s_i, there are only at most 12 additional mismatches between occurrences of symbols from V in s_i and $\star$ in s. This is a contradiction to the minimality of d'. □

Consequently, $s = \star^6 v_{p_1} \star^7 v_{p_2} \star^7 \dots v_{p_n} \star^7$ with $v_{p_i} \in N_i$, $1 \leq i \leq n$, and s is aligned with position $1, 3, 5$ or 7 of s_i, $1 \leq i \leq n$. Next, we show that $d' = d$.

Claim: $d' = d$.

Proof of Claim: Every position δ_i of s matches with the corresponding symbol of exactly one of the 4 strings q_r, $1 \leq r \leq 4$, and therefore causes 3 mismatches. All other positions $j \notin \{\delta_1, \delta_2, \dots, \delta_n\}$ are matches with respect to the strings q_r. Thus, the weight caused by the mismatches between s and the strings q_r is exactly $3(n+1)n$. This implies that a weight of at most $d - 3(n+1)n = n^2 + 8n$ is caused by mismatches between s and all strings s_i. However, the minimum number of mismatches between s and any fixed string s_i is $(n-4)+12 = n+8$, i. e., at most 4 of the n positions δ_j of s match the corresponding symbol in s_i and the other $(n-4)$ positions δ_j cause mismatches with $\star$, and the remaining 12 occurrences of symbols from V in s_i are mismatches with $\star$ in s). Hence, the minimum weight due to the strings s_i is $n(n+8) = n^2 + 8n$, too; thus, $d' = d$. □

Since $v_{p_i} \in N_i$, $1 \leq i \leq n$, every symbol v_{p_i} can be interpreted as a vertex selected from N_i. In order to conclude that these vertices $\{v_{p_1}, v_{p_2}, \dots, v_{p_n}\}$ form a perfect code, we have to show that if a vertex is select from N_i, then it must also be selected from all neighbourhoods in which it is contained, i. e., $v_{p_i} \in N_j$ implies $v_{p_i} = v_{p_j}$, which is established by the following claim.

Claim: If s is aligned with s_i at position $u \in \{1, 3, 5, 7\}$, then, for every j, $1 \leq j \leq n$, with $v_j \in N_i$, $v_{p_j} = v_{\wp_r(i)}$, where $r = \frac{u+1}{2}$.

Proof of Claim: We first assume that s is aligned with s_i at position $u = 1$. This means that, for every j, $1 \leq j \leq n$, the position δ_j of s, which carries the symbol v_{p_j}, is aligned with position δ_j of s_i. By the structure of s_i, we know that exactly the 4 positions $\delta_{j'}$ of s_i with $v_{j'} \in N_i$ carry the symbol $v_{\wp_1(i)}$, whereas all other $n-4$ positions $\delta_{j''}$ with $v_{j''} \notin N_i$ carry the symbol $\star$. In particular, this implies that there are at least $n-4$ mismatches due to the positions δ_j, $1 \leq j \leq n$, of s. Furthermore, all other 12 occurrences of symbols from V in s_i (i. e., the ones not corresponding to a position δ_j) constitute mismatches with $\star$ in s. Since the total number of mismatches between s and s_i is at most $(n-4)+12$, the 4 positions $\delta_{j'}$ of s_i with $v_{j'} \in N_i$ must be matches and therefore $s[\delta_{j'}] = v_{p_{j'}} = v_{\wp_1(i)}$. The cases $u \in \{3, 5, 7\}$ can be handled analogously; the only difference is that positions δ_j of s are aligned with positions $\delta_j + (u-1)$ of s_i. □

By the claims from above, $C = \{v_{p_1}, v_{p_2}, \dots, v_{p_n}\} \subseteq V$ with $v_{p_i} \in N_i$, $1 \leq i \leq n$.

If, for some j, $1 \leq j \leq n$, $|N_j \cap C| > 1$, then there exists an i, $1 \leq i \leq n$, $i \neq j$, such that $v_{p_i} \in N_j$ and $v_{p_j} \in N_j$ with $v_{p_i} \neq v_{p_j}$; a contradiction to the previous claim. Consequently, C is a perfect code, which concludes the proof. □

In order to complete the proof of Theorem 7, it remains to show how the alphabet size can be bounded by 5. To this end, we first slightly modify the reduction by adding substrings $\star^6$ between substrings $t_{i,j}$ and $t_{i,j+1}$ of s_i and between substrings $v_{\wp_r(j)}\star^7$ and $v_{\wp_r(j+1)}\star^7$ of q_r, i. e., $s_i = \star^6 t_{i,1}\star^6 t_{i,2}\star^6 \dots t_{i,n}\star^6$, $1 \leq i \leq n$, and $q_r = \star^6 v_{\wp_r(1)}\star^7\star^6 v_{\wp_r(2)}\star^7\star^6 \dots v_{\wp_r(n)}\star^7$, $1 \leq r \leq 4$. Furthermore, we set $m = 8n + 6 + 6(n-1) = 14n$ and $d = 4n^2 + 11n$. We note that $\ell = 8n + 12 + 6(n-1) = 14n + 6$ and therefore $(\ell - m) = 6$. This reduction is still correct (in fact, the proofs apply in the same way).

Due to the newly introduced substrings $\star^6$ of the modified reduction, for every j, $1 \leq j \leq m$, $\bigcup_{i=1}^{n} \mathrm{alph}(s_i[j..j + (|s_i| - m)]) = N_{i'} \cup \{\star\}$, for some i', $1 \leq i' \leq n$. Hence, for every j, $1 \leq j \leq m$, $|\bigcup_{i=1}^{n} \mathrm{alph}(s_i[j..j + (|s_i| - m)]) \cup \{q_r[j] \mid 1 \leq r \leq 4\}| = 5$; thus, by Lemma 1, there is an equivalent instance over an alphabet of size 5 (with $(\ell - m) = 6$), which can be computed in polynomial time.

While Theorem 7 proves the fixed-parameter intractability of ConsPat$((\ell - m), |\Sigma|)$ (assuming $\mathsf{P} \neq \mathsf{NP}$), it leaves a gap with respect to smaller constant bounds for $(\ell - m)$ and $|\Sigma|$. In order to improve the reduction with respect to $(\ell - m)$, the problem Hitting Set comes to mind, which is still NP-hard if every set has 2 elements. However, for this problem it is not clear how to cater for the size bound of the desired hitting set. We conjecture that an analogous reduction, with slightly lower $(\ell - m)$, from Negation Free 1-in-3 3Sat is possible, but the proof would be more involved, since we do not have the regularity property. The parameter $|\Sigma|$ is probably more interesting, since for applications in computational biology the alphabet size is typically 4. In this regard, the parameterised complexity of ConsPat$((\ell - m), |\Sigma| = 4)$ is still open.

References

1. L. Bulteau, F. Hüffner, C. Komusiewicz, and R. Niedermeier. Multivariate algorithmics for np-hard string problems. *EATCS Bulletin*, 114:31–73, 2014.
2. P. A. Evans, A. D. Smith, and H. T. Wareham. On the complexity of finding common approximate substrings. *Theoretical Computer Science*, 306:407–430, 2003.
3. M. R. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of motif search problems. *Combinatorica*, 26:141–167, 2006.
4. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag New York, Inc., 2006.
5. M. Frances and A. Litman. On covering problems of codes. *Theory of Computing Systems*, 30:113–119, 1997.
6. J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37:25–42, 2003.
7. J. Kratochvíl and M. Křivánek. On the computational complexity of codes in graphs. In *Proc. 13th Symposium on Mathematical Foundations of Computer Science, MFCS 1988*, volume 324 of *LNCS*, pages 396–404, 1988.
8. D. Marx. Closest substring problems with small distances. *SIAM Journal on Computing*, 38:1382–1410, 2008.