Theory of Computing Systems, Vol. 39(4), pp. 545-560, 2006

Parameterized Intractability of Distinguishing Substring Selection*

Jens Gramm† Jiong Guo‡ Rolf Niedermeier‡
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
gramm,guo,niedermr@informatik.uni-tuebingen.de

Abstract

A central question in computational biology is the design of genetic markers to distinguish between two given sets of (DNA) sequences. This question is formalized as the NP-complete Distinguishing Substring Selection problem (DSSS for short) which asks, given a set of "good" strings and a set of "bad" strings, for a solution string which is, with respect to Hamming metric, "away" from the good strings and "close" to the bad strings. More precisely, given integers d_g, d_b, and L, we ask for a length-L string s such that s has Hamming distance at least d_g to every length-L substring of the good strings and such that every bad string has a length-L substring with Hamming distance at most d_b to s.

Studying the parameterized complexity of DSSS, we show that, already for binary alphabet, DSSS is W[1]-hard with respect to its natural parameters. This, in particular, implies that a recently given polynomial-time approximation scheme (PTAS) by Deng *et al.* [6, 7] cannot be replaced by a so-called *efficient* polynomial-time approximation scheme (EPTAS) [4] unless an unlikely collapse in parameterized complexity theory occurs. This is seemingly the first computational biology problem for which such a border between PTAS (which exists) and EPTAS (which is unlikely to exist) could be established.

By way of contrast, for a special case of DSSS, we present an exact fixed-parameter algorithm solving the problem efficiently. In this way, we

*An extended abstract of this paper appeared under the title "On exact and approximation algorithms for Distinguishing Substring Selection" at the 14th International Symposium on Fundamentals of Computation Theory (FCT 2003), Springer-Verlag, LNCS 2751, pages 195–209, held in Malmö, Sweden, August 13–15, 2003.

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

‡Supported by the Deutsche Forschungsgemeinschaft (DFG), junior research group PIAF (fixed-parameter algorithms), NI 369/4.

also exhibit a sharp border between fixed-parameter tractability and intractability results.
Keywords. NP-hardness, parameterized complexity, fixed-parameter intractability, approximation algorithms, PTAS, exact algorithms, computational biology, consensus word analysis, distinguishing substring selection, marker design.

1 Introduction

Motivation. Recently, there has been strong interest in developing polynomial-time approximation schemes (PTAS's) for several string problems motivated by computational molecular biology [6, 7, 8, 19, 20]. More precisely, all these problems adhere to a scenario where we are looking for a string which is "close" to a given set of strings and, in some cases, which shall also be "far" from another given set of strings. Applications in molecular biology include motif discovery (when we search a motif common to one set of sequences which are, e.g., related to a common expression behavior, opposed to a set of sequences not exhibiting this behavior) and primer design (when we search for primers that "mark" one type of sequences opposed to a second type); see Lanctot *et al.* [18] for an overview on these kinds of applications. The underlying distance measure is Hamming metric. The list of problems in this context, for which PTAS's were given, includes Closest (Sub)String [19], Consensus Patterns [20], and Distinguishing (Sub)String Selection [6, 7, 8]. All these problems are NP-complete, hence polynomial-time exact solutions are out of reach and PTAS's might be the best one can hope for. PTAS's, however, often carry huge hidden constant factors that make them useless from a practical point of view. This difficulty also occurs with the problems mentioned above. Hence, two natural questions arise.

1. To what extent can the above approximation schemes be made really practical? [1]

2. Are there, besides pure heuristics, theoretically satisfying approaches to solve these problems exactly, perhaps based on a parameterized point of view [2, 12]?

Problem definition. In this paper, we address both these questions, focusing on the Distinguishing Substring Selection problem (DSSS):
Input: Given an alphabet Σ of constant size, two sets of strings over Σ,

[1] As Fellows [12] put it in his survey, "it would be interesting to sort out which problems with PTAS's have any hope of practical approximation". Also see the new surveys by Downey [9] and Fellows [13] for good expositions on this issue.

- $S_g = \{s_1, \ldots, s_{k_g}\}$, each string of length at least L (the "good" strings),[2]
- $S_b = \{s'_1, \ldots, s'_{k_b}\}$, each string of length at least L (the "bad" strings),

and two non-negative integers d_g and d_b.
Question: Is there a length-L string s over Σ such that

- in every $s_i \in S_g$, for *every* length-L substring t_i, $d_H(s, t_i) \geq d_g$ and
- every $s'_i \in S_b$ has *at least one* length-L substring t'_i with $d_H(s, t'_i) \leq d_b$?

Here, $d_H(s, t_i)$ denotes the Hamming distance between strings s and t_i. Following Deng *et al.* [6, 7], we distinguish DSSS from Distinguishing String Selection (DSS) in which all good and bad strings have the same length L; note that Lanctot *et al.* [18] did not make this distinction and denoted both problems as DSS.
The above mentioned Closest Substring is the special case of DSSS where the set of good strings is empty. Furthermore, Closest String is the special case of Closest Substring where all given strings and the goal string have the same length.
Previous results. Since Closest String is known to be NP-complete [15, 18], the NP-completeness of Closest Substring, DSS, and DSSS immediately follows. All the mentioned problems carry at least two natural input parameters ("distance" and "number of input strings") which often are small in practice when compared to the overall input size. This leads to the important question whether the seemingly inevitable "combinatorial explosion" in exact algorithms for these problems can be restricted to some of the parameters—this is the parameterized complexity approach [2, 9, 10, 12, 13]. A problem for which this can be successfully done allows for a *fixed-parameter algorithm* and is called *fixed-parameter tractable*. In [16], fixed-parameter algorithms are given for Closest String both for the "distance" parameter as well as the parameter "number of input strings". However, Closest String is the easiest of these problems. As to Closest Substring, fixed-parameter intractability (in the above sense of restricting combinatorial explosion to parameters) was recently shown with respect to the parameter "number of input strings" [14]. More precisely, a proof of W[1]-hardness (see [10] for details on parameterized complexity theory) was given. This result already holds for binary alphabet.
New results. In this work, we show that DSSS is fixed-parameter intractable (i.e., W[1]-hard) with respect to all natural parameters as given in the problem definition and, thus, in particular, with respect to the distance parameters.

[2]Deng *et al.* [6, 7] let all good strings be of same length L; we come back to this restriction in Section 4. The terminology "good" and "bad" has its motivation in the application [18] of designing genetic markers to distinguish the sequences of harmful germs (to which the markers should bind) from human sequences (to which the markers should not bind).

	Approximation	Parameterized Complexity w.r.t. the number of input strings	Parameterized Complexity w.r.t. the distance parameter(s)
Closest String	PTAS [19]	FPT [16]	FPT [16]
Closest Substring	PTAS [19]	W[1]-hard [14]	?
Distinguishing Substring Selection	PTAS [6]	W[1]-hard$^{(*)}$	W[1]-hard$^{(*)}$

Table 1: Overview on results concerning the approximation and the parameterized complexity of Closest (Sub)String and Distinguishing Substring Selection for alphabets of constant size. We consider the parameterized complexity with respect to the two natural parameterizations, namely the distance parameter(s) and number of input strings (FPT meaning that the problem is fixed-parameter tractable). Results from this paper are marked by $^{(*)}$, "?" indicates an open question.

Besides of the interest in its own concerning the impossibility[3] of efficient exact fixed-parameter algorithms, this result also has important consequences concerning approximation algorithms. More precisely, our result implies that no efficient polynomial-time approximation scheme (EPTAS) in the sense of Cesati and Trevisan [4] is available for DSSS. As a consequence, there is strong theoretical support for the claim that the recent PTAS of Deng *et al.* [6, 7] cannot be made practical. In addition, we indicate an instructive border between fixed-parameter tractability and fixed-parameter intractability for DSSS which, somewhat surprisingly, lies between alphabets of size two and alphabets of size greater than two. All results in this paper refer to alphabets Σ of constant size.

Approximation versus parameterized complexity. Table 1 gives an overview on results concerning the approximability and the parameterized complexity of the mentioned problems (considering the practically most relevant case of constant-size alphabets), including results from this work. Table 1 shows that the results of this paper (and of [14]) give a strong theory-based support for the common intuition that both Closest Substring and Distinguishing Substring Selection (W[1]-hard) seem to be much harder problems than Closest String (fixed-parameter tractable). This observation does not derive from results of approximation theory (all problems have a PTAS). Considering Closest Substring, it was conjectured that Closest Substring is also fixed-parameter intractable with respect to the distance parameter, but it is an open

[3]Unless an unlikely collapse in structural parameterized complexity theory occurs [12].

question to prove (or disprove) this statement;[4] notably, W[1]-hardness would imply, analogously as for DSSS, that the PTAS for Closest Substring cannot replaced by an efficient PTAS.

Structure of this work. In Section 2, we give a brief introduction to parameterized complexity and approximation schemes and provide an overview on further related work. In Section 3, we show the parameterized intractability of DSSS, the main result of this paper. In Section 4, we consider a special case of DSSS, leading to a modified problem formulation, for which we, then, present a fixed-parameter algorithm. Section 5 summarizes our findings and contains directions for future work.

2 Preliminaries and Previous Work

Parameterized complexity. Given an undirected graph $G = (V, E)$ with vertex set V, edge set E, and a positive integer k, the NP-complete Vertex Cover problem is to determine whether there is a subset of vertices $C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in C. Vertex Cover is *fixed-parameter tractable* with respect to the parameter k. There now are algorithms solving it in less than $O(1.3^k + kn)$ time [5, 21, 22], where n denotes the number of graph vertices. The corresponding complexity class is called FPT. By way of contrast, consider the NP-complete Clique problem: Given an undirected graph $G = (V, E)$ and a positive integer k, does there exist a subset of vertices $C \subseteq V$ with at least k vertices such that C forms a clique by having all possible edges between the vertices in C? Clique appears to be *fixed-parameter intractable*: It is *not* known whether it can be solved in $f(k) \cdot n^{O(1)}$ time, where f might be an arbitrarily fast growing function only depending on k.

Downey and Fellows developed a completeness program for showing fixed-parameter intractability [10]. We very briefly sketch some integral parts of this theory.

Let $L, L' \subseteq \Sigma^* \times \mathbf{N}$ be two parameterized languages.[5] For example, in the case of Clique, the first component is the input graph coded over some alphabet Σ and the second component is the positive integer k, that is, the parameter. We say that L *reduces to* L' *by a standard parameterized m-reduction* if there are functions $k \mapsto k'$ and $k \mapsto k''$ from $\mathbf{N}$ to $\mathbf{N}$ and a function $(x, k) \mapsto x'$ from $\Sigma^* \times \mathbf{N}$ to Σ^* such that

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

[4] In fact, more hardness results for *unbounded* alphabet size are known [11, 14]. Here, we refer to the practically more relevant case of constant alphabet size.

[5] Generally, the second component (representing the parameter) can also be drawn from Σ^*; for most cases (particularly, within this paper), assuming the parameter to be a positive integer (or a tuple of positive integers) is sufficient.

2. $(x,k) \in L$ iff $(x',k') \in L'$.

In the subsequent section we will present a reduction from Clique to DSSS, mapping the Clique parameter k into all *four* parameters of DSSS; i.e., k' in fact is a four-tuple $(k_g, k_b, d_g, d_b) = (1, \binom{k}{2}, k+3, k-2)$ (see Section 3.1 for details). Notably, most reductions from classical complexity turn out *not* to be parameterized ones. The basic reference degree for fixed-parameter intractability, W[1], can be defined as the class of parameterized languages that are equivalent to the Short Turing Machine Acceptance problem (also known as the k-Step Halting problem). Here, we want to determine, for an input consisting of a nondeterministic Turing machine M (with unbounded nondeterminism and alphabet size), and a string x, whether or not M has a computation path accepting x in at most k steps. This can trivially be solved in $O(n^{k+1})$ time (where n denotes a bound on the total input size) and we would be surprised if this can be much improved. Therefore, this is the parameterized analogue of the Turing Machine Acceptance problem that is the basic generic NP-complete problem in classical complexity theory, and the conjecture that FPT $\neq$ W[1] is very much analogous to the conjecture that P $\neq$ NP. Other problems that are W[1]-*hard* (and also W[1]-*complete*) include Clique and Independent Set, where the parameter is the size of the relevant vertex set [10]. W[1]-hardness gives a concrete indication that a parameterized problem with parameter k is unlikely to allow for a solving algorithm with $f(k) \cdot n^{O(1)}$ running time, i.e., restricting the combinatorial explosion to the parameter. For recent surveys we refer to [2, 9, 12, 13].

Approximation. In the following, we explain some basic terms of approximation theory, thereby restricting attention to minimization problems. Given a minimization problem, a solution of the problem is $(1+\epsilon)$-*approximate* if the cost of the solution is d, the cost of an optimal solution is d_{opt}, and $d/d_{opt} \leq 1+\epsilon$. A *polynomial-time approximation scheme (PTAS)* is an algorithm that computes, for any given real $\epsilon > 0$, a $(1+\epsilon)$-approximate solution in polynomial time wherever ϵ is considered to be (a small) constant. For an overview on approximation algorithms, refer to [3, 17, 23]. Often, PTAS's have a running time $n^{O(1/\epsilon)}$, often with large constant factors hidden in the exponent which make them infeasible already for moderate approximation ratio. Therefore, Cesati and Trevisan [4] proposed the concept of an *efficient* polynomial-time approximation scheme (EPTAS) where the PTAS is required to have $f(\epsilon) \cdot n^{O(1)}$ running time where f is an arbitrary function depending only on ϵ and not on n. Notably, seemingly most known PTAS's are *not* EPTAS's [9, 12, 13].

Previous work. Lanctot *et al.* [18] initiated the research on the algorithmic complexity of distinguishing string selection problems. In particular, besides showing NP-completeness (an independent NP-completeness result was proven by Frances and Litman [15]), they gave a polynomial-time factor-2-approximation

for DSSS. Building on PTAS algorithms for Closest String and Closest Substring [19], Deng *et al.* [8] gave PTAS's for DSS (which even holds for unbounded alphabet size), and Deng *et al.* [6, 7] recently gave a PTAS for DSSS. Concerning the running times of these PTAS's, consider, for example, the one for Closest String [19], the starting point in this series of results: For k length-L input strings, it has a worst-case bound on the running time of $O((k \cdot L)^r \cdot k^{O(\log|\Sigma| \cdot r^2/\epsilon^2)})$ in order to achieve a factor-$(1 + (1/(2r-1)) + \epsilon)$ approximation, for an integer r, $2 \leq r$, and real $\epsilon > 0$. For achieving a 25 percent error, we find the optimal trade-off between r and ϵ in this term for $r = 5$ and $\epsilon \approx 0.14$; then, the r^2/ϵ^2 term in the exponent of the running time estimation already evaluates to more than 1200. The corresponding polynomial running time is enormous and the algorithm is clearly impractical.

There appear to be no nontrivial results on exact or fixed-parameter algorithms for DSSS. Since Closest Substring is a special case of DSSS, the fixed-parameter intractability results for Closest Substring [11, 14] also apply to DSSS, implying that DSSS is W[1]-hard with respect to the parameter "number of input strings".

Concerning the special case DSS (where all given input strings have exactly the same length as the goal string) of DSSS, the fixed-parameter algorithm for Closest String with respect to the number of input strings [16] can easily be extended to solve DSS, showing that the problem is fixed-parameter tractable with respect to the aggregate parameter (k_g, k_b). It is open whether or not DSS is fixed-parameter tractable with respect to the distance parameters d_b and d_g. However, DSS is fixed-parameter tractable when considering, instead of distance parameter d_g, its "dual" parameter $d'_g = L - d_g$: DSS is solvable in $O((k_g + k_b) \cdot L \cdot (\max\{d_b + 1, (d'_g + 1) \cdot (|\Sigma| - 1)\})^{d_b})$ time [16], i.e., for constant alphabet size, it is fixed-parameter tractable with respect to the aggregate parameter (d'_g, d_b). In a sense, DSS relates to DSSS as Closest String relates to Closest Substring and, thus, DSS should be regarded as considerably easier (and of less practical importance) than DSSS.

3 Fixed-Parameter Intractability of DSSS

We show that DSSS is, even for binary alphabet, W[1]-hard with respect to the aggregate parameter (d_g, d_b, k_g, k_b). This also means hardness for every single of these parameters. With [4], this implies that DSSS does not have an EPTAS.

To simplify the presentation, we use the following technical terms. Regarding the good strings, we say that a length-L string s *matches* an $s_i \in S_g$ or, equivalently, s is a *match* for s_i, if $d_H(s, t_i) \geq d_g$ for every length-L substring t_i of s_i. Regarding the bad strings, we say that s *matches* an $s'_i \in S_b$ or, equivalently, s is a *match* for s'_i, if there is a length-L substring t'_i of s'_i with $d_H(s, t'_i) \leq d_b$. Both these notions of matching for good as well as for bad strings generalize to sets of strings

in the natural way.
Our hardness proof follows a similar structure as the W[1]-hardness proof for Closest Substring [14]. We give a parameterized reduction from Clique to DSSS. Here, however, the reduction has novel features in two ways. Firstly, from the technical point of view, the reduction becomes much more compact and, thus, more elegant. Secondly, for Closest Substring with binary alphabet, we could only show W[1]-hardness with respect to the number of input strings. Here, however, we can show W[1]-hardness with respect to, among others, parameters d_g and d_b. This has a strong consequence: Here, we can conclude that DSSS has no EPTAS, which is an open question for Closest Substring [14].

3.1 Reduction from Clique to DSSS

A Clique instance is given by an undirected graph $G = (V, E)$, with a set $V = \{v_1, v_2, \ldots, v_n\}$ of n vertices, a set E of m edges, and a positive integer k denoting the desired clique size. We describe how to generate two sets of strings over alphabet $\{0, 1\}$, S_g (containing one string s_g of length $L := nk + 5$) and S_b (containing $\binom{k}{2}$ strings, each of length $m \cdot (2nk + 5) + (m - 1)$), such that G has a clique of size k iff there is a length-L string s which is a match for S_g as well as for S_b; this means that $d_H(s, s_g) \geq d_g$ with $S_g := \{s_g\}$ and $d_g := k + 3$, and every $s'_b \in S_b$ has a length-L substring t'_b with $d_H(s, t'_b) \leq d_b$ and $d_b := k - 2$. In the following we use "$\cdot$" to denote the concatenation of strings.
Good string. $S_g := \{s_g\}$ where $s_g = 0^L$, the all-zero string of length L.
Bad strings. $S_b := \{s'_{1,2}, \ldots, s'_{1,k}, s'_{2,3}, s'_{2,4}, \ldots, s'_{k-1,k}\}$, where *every* $s'_{i,j}$ has length $m \cdot (2nk + 5) + (m - 1)$ and encodes the whole graph; in the following, we describe how we generate a string $s'_{i,j}$.
We encode a vertex $v_r \in V$, $1 \leq r \leq n$, in a length-n string by setting the rth position of this string to "1" and all other positions to "0", i.e.,

$$\langle \text{vertex}(v_r) \rangle := 0^{r-1}10^{n-r}.$$

In $s'_{i,j}$, we encode an edge $\{v_r, v_s\} \in E$, $1 \leq r < s \leq n$, by a length-(nk) string

$$\langle \text{edge}(i, j, \{v_r, v_s\}) \rangle := \underbrace{0^n \ldots 0^n}_{(i-1)} \cdot \langle \text{vertex}(v_r) \rangle \cdot \underbrace{0^n \ldots 0^n}_{(j-i-1)} \cdot \langle \text{vertex}(v_s) \rangle \cdot \underbrace{0^n \ldots 0^n}_{(k-j)}.$$

Furthermore, for purely technical reasons, we define

$$\langle \text{edge\_block}(i, j, \{v_r, v_s\}) \rangle := \langle \text{edge}(i, j, \{v_r, v_s\}) \rangle \cdot 01110 \cdot \langle \text{edge}(i, j, \{v_r, v_s\}) \rangle.$$

Before explaining why we choose this way of constructing the $\langle \text{edge\_block}(\cdot, \cdot, \cdot) \rangle$ strings, we define, given $E = \{e_1, \ldots, e_m\}$, the resulting set of bad strings by

$$s'_{i,j} := \langle \text{edge\_block}(i, j, e_1) \rangle \cdot 0 \cdot \langle \text{edge\_block}(i, j, e_2) \rangle \cdot \cdots \cdot \langle \text{edge\_block}(i, j, e_m) \rangle$$

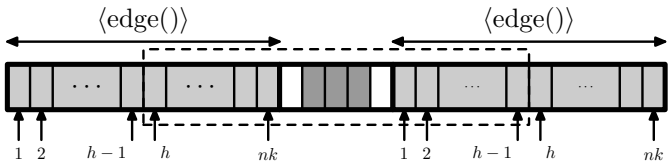

Figure 1: Illustration of Observation 1. Shown is an ⟨edge_block()⟩ string, constructed as explained in Section 3.1; boxes are used to display bits, dark-shaded boxes denote bits set to 1, white boxes denote bits set to 0. The shown string consists of two ⟨edge()⟩ strings (light-shaded, details are omitted) of length nk, separated by a "01110" string. The dashed box indicates a length-$(nk+5)$ substring which contains the "01110" substring: It necessarily starts at a position h, $1 \leq h \leq nk+1$, in the first ⟨edge()⟩ substring and extends to position $h-1$ in the second ⟨edge()⟩ substring.

for $1 \leq i < j \leq k$. Now, we give a motivation for our construction of the ⟨edge_block$(\cdot,\cdot,\cdot)$⟩ strings. Essential is the following obvious observation, where we use $s[h_1, h_2]$ to denote the substring of a string s ranging from position h_1 to position h_2, $1 \leq h_1 \leq h_2 \leq |s|$. If $h_1 > h_2$, then $s[h_1, h_2]$ denotes the empty string.

Observation 1. *Let $s' := \langle edge\_block(i,j,e)\rangle$ for positive integers i, j, $i < j$, and $e \in E$. Then, every length $L = nk+5$ substring of s' which contains the "01110" substring has the form*

$$\langle edge(i,j,e)\rangle[h, nk] \cdot 01110 \cdot \langle edge(i,j,e)\rangle[1, h-1]$$

for $1 \leq h \leq nk+1$. □

An illustration of Observation 1 is given in Fig. 1. This observation will be useful for the following reason. The goal of our construction is that, assuming a solution string s of the produced DSSS instance, a match for s in a bad string $s'_{i,j}$ contains all "information", i.e., all bits, of one of the ⟨edge(i,j,e)⟩ substrings encoded in $s'_{i,j}$ for some $e \in E$. This is achieved by using Observation 1 since, as will be shown in Section 3.2, it is possible to enforce that a match contains the "01110" substring.

Parameter values. We set $L := nk+5$ and generate $k_g := 1$ good string, $k_b := \binom{k}{2}$ bad strings, and we set distance parameters $d_g := k+3$ and $d_b := k-2$.

Example. Let $G = (V, E)$ with $V := \{v_1, v_2, v_3, v_4\}$ and $E := \{\{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_3, v_4\}\}$ as shown in Fig. 2(a) and let $k = 3$. Fig. 2(b) displays the good string s_g and the $\binom{k}{2} = 3$ bad strings $s'_{1,2}$, $s'_{1,3}$, and $s'_{2,3}$. Additionally, we show the length-$(nk+5)$, i.e. length-17, string s which is a match for $S_g = \{s_g\}$ and a match for $S_b = \{s'_{1,2}, s'_{1,3}, s'_{2,3}\}$ and, thus, corresponds to the k-clique in G.

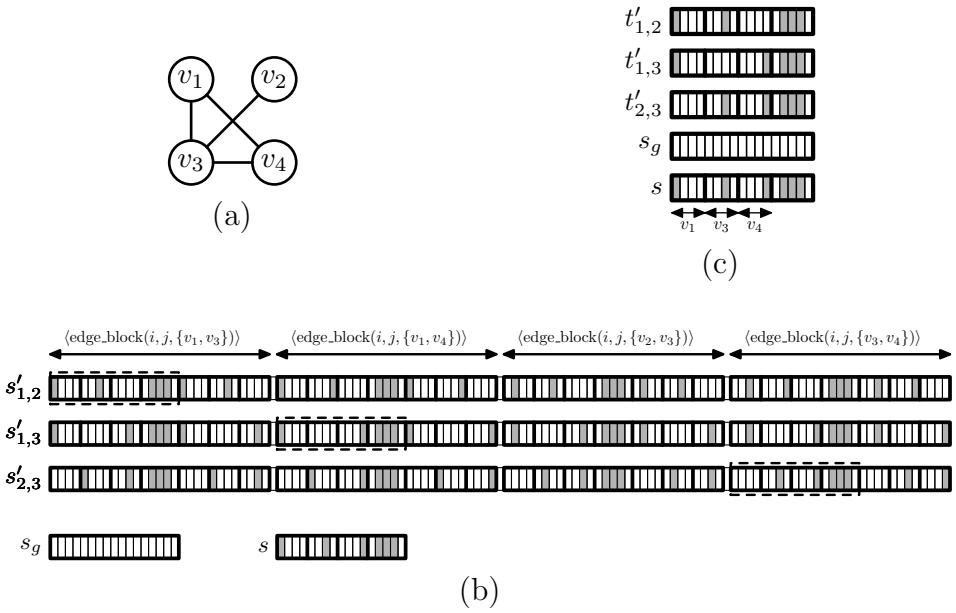

Figure 2: Example for the reduction from a Clique instance to a DSSS instance with binary alphabet. (a) A Clique instance $G = (V, E)$ with $k = 3$. (b) The produced DSSS instance. We indicate the "1"s of the construction by grey boxes, the "0"s by white boxes. We display the solution s that is found since G has a clique of size $k = 3$; matches of s in $s'_{1,2}$, $s'_{1,3}$, and $s'_{2,3}$ are indicated by dashed boxes. By bold lines we indicate the substrings by which we constructed the bad strings: each $\langle \text{edge\_block}(\cdot, \cdot, e)\rangle$ substring is built from $\langle \text{edge}(\cdot, \cdot, e)\rangle$ for some $e \in E$, consisting of k length-n substrings, followed by "01110", followed again by $\langle \text{edge}(\cdot, \cdot, e)\rangle$. (c) Alignment of the matches $t'_{1,2}$, $t'_{1,3}$, and $t'_{2,3}$ (marked by dashed boxes in (b)) with s_g and s.

3.2 Correctness of the Reduction

We show the two directions of the correctness proof for the above construction by two lemmas.

Lemma 1. *For a graph with a k-clique, the construction in Section 3.1 produces an instance of* DSSS *that has a solution, i.e., there is a length-L string s such that $d_H(s, s_g) \geq d_g$ and every $s'_{i,j} \in S_b$ has a length-L substring $t'_{i,j}$ with $d_H(s, t'_{i,j}) \leq d_b$.*

Proof. Let $h_1, h_2, \ldots, h_k$ denote the indices of the clique's vertices, $1 \leq h_1 < h_2 < \cdots < h_k \leq n$. Then, we can find a solution string

$$s := \langle \text{vertex}(v_{h_1})\rangle \cdot \langle \text{vertex}(v_{h_2})\rangle \cdot \cdots \cdot \langle \text{vertex}(v_{h_k})\rangle \cdot 01110.$$

For every $s'_{i,j}$, $1 \leq i < j \leq k$, the bad string $s'_{i,j}$ contains a substring $t'_{i,j}$ with

$d_H(s, t'_{i,j}) = d_b = k - 2$, namely

$$t'_{i,j} := \langle \text{edge}(i, j, \{v_{h_i}, v_{h_j}\}) \rangle \cdot 01110.$$

Moreover, we have $d_H(s, s_g) = d_g = k + 3$. □

Lemma 2. *A solution for the* DSSS *instance produced from a graph G by the construction in Section 3.1 corresponds to a k-clique in G.*

Proof. We prove this statement in several steps:
(1) We observe that a solution for the DSSS instance has at least $k + 3$ "1"s since $d_H(s, s_g) \geq d_g = k + 3$ and s_g is the all-zero string.
(2) We observe that a solution for the DSSS instance has at most $k+3$ many "1"s: Following the construction, every length-L substring $t'_{i,j}$ of every bad string $s'_{i,j}$, $1 \leq i < j \leq k$, contains at most five "1"s and $d_H(s, t'_{i,j}) \leq k - 2$.
(3) A match $t'_{i,j}$ for s in the bad string $s'_{i,j}$ contains exactly five "1"s: This follows from the observation that *every* length-L substring in a bad string contains *at most* five "1"s together with (1) and (2): Only if $t'_{i,j}$ contains five "1"s and all of them coincide with "1"s in s, we have $d_H(s, t'_{i,j}) = (k + 3) - 5 = k - 2$.
(4) All $t'_{i,j}$, $1 \leq i < j \leq k$, and s must contain a "111" substring, located at the same position: To show this, let $t'_{i,j}$ be a match of s in a bad string $s'_{i,j}$ for some $1 \leq i < j \leq k$. From (3), we know that the match $t'_{i,j}$ must contain exactly five "1"s. Thus, since a substring of a bad string contains five "1"s only if it contains a "111" substring, $t'_{i,j}$ must also contain a "111" substring (which separates in $s'_{i,j}$ two substrings $\langle \text{edge}(i, j, e) \rangle$ for some $e \in E$). All "1"s in $t'_{i,j}$ have to coincide with "1"s chosen from the $k+3$ "1"s in s. In particular, the position of the "111" substring must be the same in s and in $t'_{i,j}$ for all $1 \leq i < j \leq k$. This ensures a "synchronization" of the matches.
(5) W.l.o.g., all $t'_{i,j}$, $1 \leq i < j \leq k$, and s end with the "01110" substring: From (4), we know that all $t'_{i,j}$ contain a "111" substring at the same position. If they do not all end with "01110", we can shift them such that the contained "111" substring is shifted to the appropriate position, as we describe more precisely in the following. Recall that every length-L substring which contains the "111" substring of $\langle \text{edge\_block}(i, j, e) \rangle$ has the form $\langle \text{edge}(i, j, e) \rangle[h, nk] \cdot 01110 \cdot \langle \text{edge}(i, j, e) \rangle[1, h - 1]$ for $1 \leq h \leq nk + 1$ and $e \in E$. Since all $t'_{i,j}$, $1 \leq i < j \leq k$, contain the "111" substring at the same position, they all have this form for the same h. Then, we can, instead, consider $\langle \text{edge}(i, j, e) \rangle[1, nk] \cdot 01110$ and, by a circular shift, move the "111" substring in the solution to the appropriate position. Considering the solution s and the matches $t'_{i,j}$ for all $1 \leq i < j \leq k$ as a character matrix, this is a reordering of columns and, thus, the pairwise Hamming distances do not change.
(6) We divide the starting nk positions of the matches and the solution into k "sections", each of length n. In s, each of these sections has the form $\langle \text{vertex}(v) \rangle$ for a vertex $v \in V$ by the following argument: By (5), all matches in bad strings

end with "01110" and, by the way we constructed the bad strings, each of their sections either consists only of "0"s or has the form $\langle \text{vertex}(v) \rangle$ for a vertex $v \in V$. If the section encodes a vertex, it contains one "1" which has to coincide with a "1" in s. For the ith section, $1 \leq i \leq k$, the matches in strings $s'_{i,j}$ for $i < j \leq k$ and in strings $s'_{j,i}$ for $1 \leq j < i$, encode a vertex in their ith section. Therefore, every of the k sections in s contains a "1" and, since s (by (1) and (2)) contains $k+3$ many "1"s and (by (4)) ends with "01110", each of its sections contains exactly one "1". Therefore, every section of s can be read as the encoding $\langle \text{vertex}(v) \rangle$ for a $v \in V$.

Summary. Following (6), let v_{h_i}, $1 \leq i \leq k$, be the vertex encoded in the ith length-n section of s. Now, consider some $1 \leq i < j \leq k$. Solution s has a match in $s'_{i,j}$ iff there is an $\langle \text{edge}(i, j, \{v_{h_i}, v_{h_j}\}) \rangle \cdot 01110$ substring in $s'_{i,j}$ and this holds iff $\{v_{h_i}, v_{h_j}\} \in E$. Since this is true for all $1 \leq i < j \leq k$, all v_{h_1}, $v_{h_2}, \ldots, v_{h_k}$ are pairwisely connected by edges in G and, thus, form a k-clique. □

Lemmas 1 and 2 yield the following theorem.

Theorem 1. DSSS *with binary alphabet is* W[1]*-hard for every combination of the parameters* k_g, k_b, d_g, *and* d_b.[6] □

Theorem 1 means, in particular, that DSSS with binary alphabet is W[1]-hard with respect to every single parameter k_g, k_b, d_g, and d_b. Moreover, it allows us to exploit an important connection between parameterized complexity and the theory of approximation algorithms as follows.

Corollary 1. *There is no EPTAS for* DSSS *unless* W[1] = FPT.

Proof. Cesati and Trevisan [4] have shown that a problem with an EPTAS is fixed-parameter tractable with respect to the parameters that correspond to the objective functions of the EPTAS. In Theorem 1, as a special case, we have shown W[1]-hardness for DSSS with respect to d_g and d_b. Therefore, we conclude that DSSS cannot have an EPTAS for the objective functions d_g and d_b unless W[1] = FPT. □

4 Exploring the Border Between Parameterized Tractability and Intractability

In the previous section, we showed that the prospects for provable efficient and exact algorithmic solvability of DSSS are rather bad. On the one hand, we showed that DSSS in W[1]-hard with respect to seemingly all natural parameters. On the other hand, we know from [16] that Closest String—the special

[6] Note that this is the strongest statement possible for these parameters because it means that the combinatorial explosion cannot be restricted to a function $f(k_g, k_b, d_g, d_b)$.

case of DSSS where all strings have the same length and the set of good strings is empty—is fixed-parameter tractable with respect to both its natural parameterizations, i.e., distance parameter and number of input strings. This naturally leads to the goal of better understanding what the reasons are that make these NP-complete problems, both having a PTAS, so different from a parameterized (exact) point of view. In a generalized meaning, here we attempt to better understand and explore the border between parameterized tractability and intractability. Surprisingly, our subsequent investigations show that the alphabet size—here not only the distinction between constant-size and unbounded-size alphabet (as usually studied) but the distinction between binary and non-binary alphabet matters—and the parameter choice seem to strongly influence the parameterized complexity. To this end, we consider a somewhat contrived, modified version of DSSS. More precisely, we consider a special case of DSSS which, then, leads to a modified problem formulation. Note, however, that it is unclear whether or not this modified version may be useful in the discussed biological application.
We impose the following restrictions on DSSS: First of all, we use, for the sake of simplicity, a "harmless" restriction already employed, e.g., by Deng *et al.* [6, 7]:

Length of good strings: We assume that all strings in S_g are of length L. Increasing the number of good strings by a factor linear in the string lengths, we can easily transform an instance of DSSS into one in which all good strings have the same length L: We replace every $s_i \in S_g$ by a set consisting of all length-L substrings of s_i.

The "real" restrictions are as follows:

Binary alphabet: We restrict the problem to a binary alphabet $\Sigma = \{0, 1\}$; this applies when the given strings encode binary properties like, e.g., the distinction between purines and pyrimidines in DNA sequences, or between hydrophobic and hydrophilic amino acids in protein sequences.

Dual parameter: We consider, instead of the parameter d_g from the DSSS definition, the "dual parameter" $d'_g := L - d_g$ such that we require a solution string s with $d_H(s, s_i) \geq L - d'_g$ for every $s_i \in S_g$. The idea behind is that in some practical cases it might occur that, while d_g is rather large, d'_g is fairly small. Hence, restricting the combinatorial explosion to d'_g might sometimes be even more natural than restricting it to d_g.[7]

Optimality of distance parameter: We require that the distance parameter d'_g is "optimal"; here, d'_g is optimal if d'_g is minimum among all values such

[7] Observe that the distance value d_g concerning the good strings is a maximization parameter whereas the corresponding value d_b for the bad strings is a minimization parameter. Hence, replacing d_g by d'_g makes DSSS for both d'_g and d_b a minimization problem which seems more natural in the light of the fact that most NP-hard problems that possess efficient fixed-parameter algorithms are minimization problems.

that there is at least one length-L string s with $d_H(s, s_i) \geq L - d'_g$ for every $s_i \in S_g$.

The restrictions outlined above lead us to a modified version of DSSS to which we refer as MDSSS:

Input: Given an alphabet $\Sigma = \{0, 1\}$ and two sets of strings over Σ,

- $S_g = \{s_1, \ldots, s_{k_g}\}$, each string of length L (the "good" strings),
- $S_b = \{s'_1, \ldots, s'_{k_b}\}$, each string of length at least L (the "bad" strings),

and a non-negative integer d_b.
Question: Is there a length-L string s such that, for an optimal value of distance parameter d'_g, $d_H(s, s_i) \geq L - d'_g$ for every $s_i \in S_g$ and such that every $s'_i \in S_b$ has a length-L substring t'_i with $d_H(s, t'_i) \leq d_b$?

Note that, in this definition, we can read the set S_g of k_g length-L strings as a $k_g \times L$ character matrix. We call a column in this matrix *dirty* if it contains "0"s as well as "1"s.
We give, in the remaining section, a fixed-parameter algorithm that solves MDSSS and, moreover, that determines the minimal d_b that allows a solution together with all solution strings corresponding to this found optimal d_b. We conclude this section by pointing out the difficulties that arise when we give up some of the restrictions concerning MDSSS. Altogether, in this way we hope to shed light on the reasons for the "parameterized (resp. exact) gap" between DSSS and Closest String.

Fixed-Parameter Algorithm for MDSSS.
The structure of the algorithm is as follows.
Preprocessing: For an all-"0" (all-"1") column in the $k_g \times L$ character matrix, we set the corresponding character in the solution string to its inverse "1" ("0"); otherwise, we cannot find a solution with an *optimal* d'_g. Columns containing "0" as well as "1" are called *dirty*. If there are more than $d'_g \cdot k_g$ dirty columns then we reject the input instance. Otherwise, we proceed on the thereby reduced set S_g consisting only of dirty columns.
The correctness of the preprocessing follows in an easy way similar to the correctness of the "problem kernel" for Closest String observed by Evans *et al.* [11]. The preprocessing can easily be done in $O(k_g \cdot L)$ time.
Phase 1: We determine *all* solution strings s such that $d_H(s, s_i) \geq L - d'_g$ for every $s_i \in S_g$ for the optimal d'_g. This phase extends the ideas behind a bounded search tree algorithm for Closest String described in [16]. There, however, the focus was on finding *one* solution string whereas, here, we require to find *all* solution strings for the optimal parameter value. This extension was only discussed in [16] and it will be described in more detail here.

The precondition of this phase is an optimal parameter d'_g. Since, in general, the optimal d'_g is not known in advance, it can be found by looping through $d'_g = 0, 1, 2, \ldots$, each time invoking the procedure described in the following until we meet the optimal d'_g. For each such d'_g value, we do not have to redo the preprocessing, but only compare the number of dirty columns against $d'_g \cdot k_g$.

Phase 1 is realized as a recursive procedure and is invoked with parameters $s_c := \text{inv}(s_1)$ for $s_1 \in S_g$ and d'_g, where $\text{inv}(s_1)$ denotes the bitwise complement of s_1.[8] The procedure distinguishes between two cases: Either (1) s_c is already a solution string for this phase or (2) it is not. In both cases, the procedure invokes recursive calls with a modified s_c and a decreased d'_g as parameters, described as follows.

In case (1), s_c is already far away from all strings in S_g (i.e., $d_H(s_c, s_i) \geq L - d'_g$ for all $s_i \in S_g$). Then, the procedure returns s_c as a solution string. Note that it is possible that s_c can be further transformed into another solution. Thus, we select a string $s_i \in S_g$ such that s_c is not allowed to be closer to s_i (i.e., $d_H(s_c, s_i) = L - d'_g$); such an s_i must exist since parameter d'_g is optimal. We try all possible ways to move s_c away from s_i (such that $d_H(s_c, s_i) = L - (d'_g - 1)$), and call the recursive procedure for each of the produced instances. Since s_i coincides with s_c in exactly d'_g positions, the procedure produces at most d'_g new instances.

In case (2), s_c is not a solution string. We select a string $s_i \in S_g$ such that s_c is too close to s_i (i.e., $d_H(s_c, s_i) < L - d'_g$); we try all possible ways to move s_c away from s_i, and we call the recursive procedure for each of the produced instances. As shown in [16], there are at most $d'_g + 1$ new instances and, thus, at most $d'_g + 1$ recursive calls.

The invocations of the recursive procedure can, thus, be described by a search tree. In the above recursive calls, we go without those calls that try to change a position in s_c which has already been changed before. We also omit further invocations of the recursive procedure if the current node of the search tree is already at depth d'_g of the tree; otherwise, s_c would move too close to s_1 (i.e., $d_H(s_c, s_1) < L - d'_g$).

The proof of the correctness of Phase 1 can be done in a similar way as in [16]. Furthermore, it is easy to observe that the search tree hat a size of $O((d'_g+1)^{d'_g}) = O((d'_g)^{d'_g})$. In each node of the search tree, we need at most $O(d'_g \cdot k_g)$ time to determine whether or not s_c is a match for S_g and to identify the positions in which s_c coincides with an $s_i \in S_g$. Together with the loop $d'_g = 0, 1, 2, \ldots$ in order to find the optimal d'_g, Phase 1 can be done in $O((d'_g)^2 \cdot k_g \cdot (d'_g)^{d'_g})$ time.

Phase 2: For every s found in Phase 1, we determine the minimal value of d_b such that every $s'_i \in S_b$ has a length-L substring t'_i with $d_H(s, t'_i) \leq d_b$. Finally, find the minimal value of d_b over all examined choices of s. For a given solution string s from the first phase and a string $s'_i \in S_b$, we use Abrahamson's algorithm [1] to find the minimum number of mismatches between s and every length-L substring

[8] The choice of s_1 was arbitrary—we could choose any of the good input strings.

of s'_i in $O(|s'_i|\sqrt{L \log L})$ time. We denote this minimum by $\min_{t'_i} d_H(s, t'_i)$, where t'_i is a length-L substring of s'_i. Applying this algorithm to all strings in S_b, we get $d_{b,s} := \max_{i=1,\ldots,k_b} \min_{t'_i} d_H(s, t'_i)$. The minimum value of $d_{b,s}$, over all solution strings s from the first phase, is then the minimum distance of a solution string from Phase 1 to all bad strings, and an s which achieves this minimum distance is a corresponding solution string.
This phase is apparently correct and can be done in $O(N\sqrt{L \log L} \cdot (d'_g)^{d'_g})$ time. Altogether, we obtain the following theorem:

Theorem 2. *MDSSS can be solved in* $O(L \cdot k_g + ((d'_g)^2 k_g + N\sqrt{L \log L}) \cdot (d'_g)^{d'_g})$ *time where* $N = \sum_{s'_i \in S_b} |s'_i|$ *is the total size of the bad strings.*

Discussion. The special requirements imposed on the input of MDSSS seem inevitable in order to obtain the above fixed-parameter tractability result. We discuss the problems arising when relaxing the constraints on the alphabet size and the value of d'_g.
Non-binary alphabet. Already extending the alphabet size in the problem formulation from two to three makes MDSSS combinatorially much more difficult such that the approach described above does not yield fixed-parameter tractability any more. A reason lies in the preprocessing. When having an all-equal column in the character matrix induced by S_g, for a three-letter alphabet there are two instead of one possible choices for the corresponding position in the solution string. Therefore, to enumerate all solutions s with $d_H(s, s_i) \geq L - d'_g$ for all $s_i \in S_g$, which is essential for our approach, is not fixed-parameter tractable: the number of solutions is too large. Let $L' \leq L$ be the number of non-dirty columns and let the alphabet size be three. Then, aside from the dirty columns, we already have $2^{L'}$ assignments of characters to the positions corresponding to non-dirty columns.
Non-optimal d'_g parameter. Also for non-optimal d'_g parameter, the number of solutions s with $d_H(s, s_i) \geq L - d'_g$ for all $s_i \in S_g$ can become too large and it appears to be fixed-parameter intractable with respect to d'_g to enumerate them all. Consider the example where $S_g = \{0^L\}$. Then, there are more than $\binom{L}{d'_g}$ strings s with $d_H(s, 0^L) \geq L - d'_g$.

5 Conclusion

Summary of Results. The main result shown in this paper is that Distinguishing Substring Selection is W[1]-hard with respect to all its natural parameters (distance values and input string numbers). Thus, the status of this problem is, generally speaking, hopeless concerning the existence of useful fixed-parameter algorithms. Moreover, our result also implies that DSSS has no efficient PTAS unless FPT = W[1], thus, bringing also bad news concerning

approximation algorithms. A way out of such a misery is to consider relevant special cases of the given problem. We did this by studying MDSSS, which restricts DSSS to binary alphabet, employs a dual "distance parameterization" for good strings, and also requires this new parameter d'_g to be optimal. We showed that MDSSS is fixed-parameter tractable with respect to the parameter d'_g. Surprisingly, the border line between (fixed-parameter) tractability and intractability seems to lie between alphabet sizes two and three.

Discussion. So far, there has been quite a lot of work on PTAS's for various NP-hard problems from computational biology (e.g., cf. [6, 7, 19, 20]) and many other fields. As Downey [9] and Fellows [12, 13] discussed in their recent surveys, the practical usefulness of many PTAS results is doubtful. This work brings one of the so far few (if any) examples with provable non-existence (unless FPT = W[1]) of an efficient PTAS. In addition, we also revealed that parameterized complexity studies here also seem of little help for practically solving the problem. Altogether, since DSSS needs to be solved in practice, this gives strong theory-based support to investigations on heuristic solutions and for the determination of practically relevant and tractable special cases of DSSS.

Open Questions and Future Work. It remains open whether analogous parameterized hardness results (in particular, with respect to distance parameters) can be shown for the closely related problems Closest Substring and Consensus Patterns [19, 20]. Both these problems have PTAS's, but the existence of EPTAS's is open. It only has been shown that, already for binary alphabet, Closest Substring and Consensus Patterns are W[1]-hard with respect to the number of input strings [14]. Furthermore, it is an issue of future research to find out whether MDSSS is of practical use in the computational biology context or whether the results on MDSSS can be further extended and improved. Finally, we conjecture that DSSS is also W[1]-hard with respect to the dual parameter $d'_g := L - d_g$ (as we considered for MDSSS), but it remains open to prove or disprove this statement.

References

[1] K. Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16(6):1039–1051, 1987.

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

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation—Combinatorial Optimization Problems and Their Approximability Properties.* Springer, 1999.

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

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

[6] X. Deng, G. Li, Z. Li, B. Ma, and L. Wang. A PTAS for Distinguishing (Sub)string Selection. In *Proc. of 29th ICALP*, volume 2380 in LNCS, pages 740–751, 2002. Springer.

[7] X. Deng, G. Li, Z. Li, B. Ma, and L. Wang. Genetic design of drugs without side-effects. *SIAM Journal on Computing*, 32(4):1073–1090, 2003.

[8] X. Deng, G. Li, and L. Wang. Center and distinguisher for strings with unbounded alphabet. *Journal of Combinatorial Optimization*, 6:383–400, 2002.

[9] R. G. Downey. Parameterized complexity for the skeptic (invited paper). In *Proc. of 18th IEEE Conference on Computational Complexity*, pages 147–169, 2003.

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

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

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

[13] M. R. Fellows. New directions and new challenges in algorithm design and complexity, parameterized. (invited paper) In *Proc. of 8th WADS*, volume 2748 in LNCS, 2003. Springer.

[14] M. R. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of Closest Substring and related problems. In *Proc. of 19th STACS*, volume 2285 in LNCS, pages 262–273, 2002. Springer.

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

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

[17] D. S. Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Company, 1997.

[18] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, 185:41–55, 2003.

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

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

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

[22] R. Niedermeier and P. Rossmanith. On efficient fixed-parameter algorithms for Weighted Vertex Cover. *Journal of Algorithms*, 47:63–77, 2003.

[23] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.