Multivariate Algorithmics for NP-Hard String Problems

Laurent Bulteau* Falk Hüffner† Christian Komusiewicz
Rolf Niedermeier

Institut für Softwaretechnik und Theoretische Informatik,
TU Berlin, Germany
l.bulteau@campus.tu-berlin.de
{falk.hueffner, christian.komusiewicz, rolf.niedermeier}@tu-berlin.de

Abstract

String problems arise in various applications ranging from text mining to biological sequence analysis. Many string problems are NP-hard. This motivates the search for (fixed-parameter) tractable special cases of these problems. We survey parameterized and multivariate algorithmics results for NP-hard string problems and identify challenges for future research.

1 Introduction

Parameterized and its sequel multivariate algorithmics strive for a fine-grained complexity analysis of NP-hard problems, with the hope to spot provably tractable cases. To this end, one analyzes how problem- and data-specific parameters influence the computational complexity of the considered problems [69, 80, 138]. So far problems from algorithmic graph theory are the main driving force for the development of the field. Areas such as computational geometry [86], computational social choice [21, 39], scheduling [23, 130, 134], or string processing still lead a comparatively quiet life in this research community. With this article, we aim to stimulate more research on NP-hard string problems using tools of parameterized and multivariate complexity analysis.

*Supported by the Alexander von Humboldt Foundation, Bonn, Germany.
†Supported by DFG project ALEPH (HU 2139/1).

String problems appear in various areas of algorithmic biology, but also in fields such as text processing, language theory, and coding theory. Notably, string problems typically come along with several natural parameters such as size of the alphabet, number of input strings, or some distance bound. Hence, it is natural to perform a *multivariate* complexity analysis [76, 116, 139]. In this context, we present numerous NP-hard string problems, discuss known results in terms of multivariate (exact) algorithmics, and feature some challenging open research questions. To keep this overview focused and within reasonable dimensions, we mostly discuss unweighted, plain string problems.

This article is organized as follows. After introducing basic concepts and notation in Section 2, we discuss consensus string problems in Section 3, common sub- and superstructure problems in Section 4, distance computation problems in Section 5, and miscellaneous NP-hard string problems in Section 6. Each section contains concrete challenges for future research. We conclude with some general remarks concerning potential directions for future work on NP-hard string problems.

2 Preliminaries

Parameterized complexity basics. Parameterized algorithmics tries to analyze problem difficulty not only in terms of the input size, but also for an additional parameter, typically an integer p. Thus, formally, an instance of a parameterized problem is a tuple of the unparameterized instance I and the parameter p. A parameterized problem with parameter p is *fixed-parameter tractable* if there is an algorithm that decides an instance (I, p) in $f(p) \cdot |I|^{O(1)}$ time, where f is an arbitrary computable function depending only on p. The complexity class that contains the fixed-parameter tractable problems is called FPT. Clearly, if the problem is NP-hard, we must expect f to grow superpolynomially. To concentrate on the contribution of f to the running time, we sometimes use the $O^*()$ notation, which omits the running time part that is polynomial in the input size.

There are parameterized problems for which there is good evidence that they are not fixed-parameter tractable. Analogously to the concept of NP-hardness, the concept of W[1]-hardness was developed. It is widely assumed that a W[1]-hard problem cannot have a fixed-parameter algorithm (hardness for the classes W[t], $t \geq 2$, has the same implication). To show that a problem is W[1]-hard, a *parameterized reduction* from a known W[1]-hard problem can be used. This is a reduction that runs in $f(p) \cdot |I|^{O(1)}$ time and maps the parameter p to a new parameter p' which is bounded by some function $g(p)$. If g is linear, that is, $p' = O(p)$, then the reduction is called *linear parameterized*

reduction.

While parameterized reductions can show that a problem is unlikely to be fixed-parameter tractable, even tighter running time lower bounds can be achieved by assuming the *Exponential Time Hypothesis* (ETH) [109]. The ETH states that the 3-SAT problem cannot be solved in $2^{o(n)}$ time, where n is the number of variables. By combining the ETH and linear parameterized reductions, one can obtain tight hardness results [48]. More precisely, if there is a linear parameterized reduction from Clique parameterized by solution size to a parameterized problem L and the ETH holds, then L cannot be solved in $|I|^{o(p)}$ time. Similarly, if there is a linear parameterized reduction from Dominating Set parameterized by solution size to L and W[1] $\neq$ FPT, then L cannot be solved in $|I|^{o(p)}$ time. For a survey on ETH-based running time lower bounds, refer to Lokshtanov et al. [122].

The notion of a *problem kernel* tries to capture the existence of provably effective preprocessing rules. More precisely, we say that a parameterized problem has a problem kernel if every instance can be reduced in polynomial time to an equivalent instance whose size depends only on the parameter. It can be shown that a problem is fixed-parameter tractable if and only if it has a problem kernel. However, the kernel derived from this might be impractically large; of particular interest are kernelizations where the size of the reduced instance depends only polynomially on the parameter. There are techniques that allow to show that a problem does not have a polynomial kernel (unless NP $\subseteq$ coNP/poly) [69, 117].

Since single parameters often lead only to intractability, it makes sense to look at *combined* parameters. For example, Longest Common Subsequence is W[2]-hard for the parameter "solution string length", but fixed-parameter tractable if additionally the alphabet size is a parameter. Depending on the application, different parameter combinations might make sense. Thus, the goal is to explore the "parameter ecology" [76, 116] of the problem and delineate the border between tractability and intractability.

Notation. Most problems we consider take one or more strings as input, and have one string as solution, which sometimes needs to fulfill some distance condition. We use the following notation for the most relevant problem parameters:

- $|\Sigma|$: alphabet size;
- k: number of input strings;
- ℓ: maximum length of an input string;

- occ: the maximum number of occurrences of any letter in the set S of input strings, that is, $\text{occ} := \max_{s \in S} \max_{a \in \Sigma} \text{occ}(a, s)$ where $\text{occ}(a, s)$ is how often letter a occurs in string s;
- m: solution string length;
- d: solution string distance.

We use the terms *substring* and *subsequence* in the standard way, that is, a substring must contain consecutive letters, while a subsequence may not. A *p-sequence* is a string in which no letter appears twice. A *permutation* is a p-sequence using all letters of the alphabet, so it has length $|\Sigma|$. The *Hamming distance* of two strings with equal length is the number of positions in which they differ. We denote by $s[i]$ the letter which is at position i of the string s.

3 Consensus Strings

In this section, we discuss consensus or median string problems. Here one searches for a string that best represents a given set of strings. We will in particular discuss the NP-complete problems Closest String (Section 3.1) and Closest Substring (Section 3.2). The main parameters are the number of strings k, the maximum input string length ℓ, the alphabet size $|\Sigma|$, the allowed maximum distance d of the solution string from the input strings, and (in the case of Closest Substring and related problems) the length m of the solution string.

3.1 Closest String

Closest String is perhaps the most basic NP-complete consensus string problem, with many applications in biology (motif search) [141, Section 8.6] but also in coding theory (minimum radius problem) [81].

Closest String
Instance: A set of k length-ℓ strings $s_1, \ldots, s_k$ over an alphabet Σ and a positive integer d.
Question: Is there a length-ℓ string $s \in \Sigma^*$ that has Hamming distance at most d to each of $s_1, \ldots, s_k$?

Figure 1 shows an example of an input instance of Closest String and its solution. Note that $m = \ell$ for this problem. Closest String is NP-complete even for binary alphabet [81]. Thus, there is no hope to

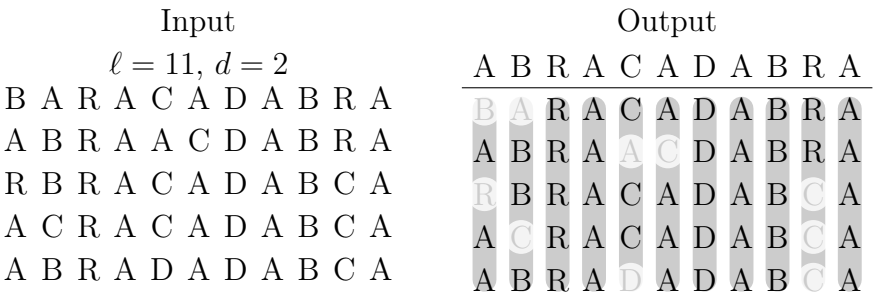

Figure 1: Closest String

obtain fixed-parameter tractability for the single parameter alphabet size $|\Sigma|$. A straightforward enumerative approach (just trying all candidate closest strings) has running time $O^*(|\Sigma|^\ell)$. This fixed-parameter tractability for the combined parameter $|\Sigma|$ and ℓ yields feasible running times only for small alphabets and short strings. Two obvious parameters, which are often small in real-world applications, are the number k of input strings and the maximum Hamming distance d, also referred to as *radius*. Using integer linear programming results [82, 115, 119], fixed-parameter tractability with respect to k can be derived [94]. This result is of purely theoretical interest due to a huge combinatorial explosion. Therefore, there have been efforts towards developing direct combinatorial algorithms for constant values of k [10, 36, 93], but a combinatorial *fixed-parameter* algorithm for parameter k is unknown.

Challenge 1. *Is there a direct combinatorial fixed-parameter algorithm for* Closest String *parameterized by the number k of input strings (thus avoiding integer linear programming)?*

The parameter d seems currently most promising in terms of obtaining practical fixed-parameter tractability results for Closest String. A simple search tree strategy basically employs the following idea: Assume that there exists a closest string with maximum distance d from the input strings. Then it must be possible to reach it from any of the input strings by changing at most d letter positions. The corresponding search can be organized in a tree-like fashion as follows: Choose any input string as a candidate closest string. As long as this candidate string has distance more than d to at least one input string, branch the search into $d + 1$ cases where in each case one picks a position in which the strings differ and changes the letter in the candidate string to the letter of the input string. This is repeated at most d times and it can be shown that if a closest string with maximum Hamming distance d exists, then this procedure finds it [94]. Altogether, this

leads to an $O^*((d+1)^d)$-time algorithm for Closest String. For small alphabets, this algorithm has been improved by employing $|\Sigma|$ as a second parameter [54, 55, 152], achieving running times of the form $O^*(|\Sigma|^{O(d)})$.

While most results in the literature care about fixed-parameter tractability versus W-hardness (parameterized intractability), Lokshtanov et al. [121], provided concrete lower bounds for algorithm running times. Assuming ETH, they showed that there is no $d^{o(d)} \cdot (k \cdot \ell)^{O(1)}$-time and no $|\Sigma|^{o(d)} \cdot (k \cdot \ell)^{O(1)}$-time algorithm for Closest String. Thus, assuming ETH, the above-mentioned algorithms are basically optimal.

In applications, Closest String is often attacked using (Integer) Linear Programming [9, 58, 154]. Exact solutions (using Integer Linear Programming) seem practically feasible for small input lengths ℓ. Using (relaxed and more efficient) Linear Programming, one only may hope for approximate solutions but one can obtain lower bounds for the Hamming distance d of an optimal solution string. In the spirit of previous work for the Vertex Cover problem [136] and the idea of parameterizing above guarantee [125]—the new (and potentially much smaller—thus stronger) parameter then is the absolute value of the difference between the LP lower bound (which can be computed in polynomial time) and the actual distance value—this leads to the following.

Challenge 2. *What is the complexity of* Closest String *parameterized above the Linear Programming (LP) relaxation of an Integer Linear Programming formulation of the problem?*

Finally, we mention in passing that Chen et al. [56] developed randomized fixed-parameter algorithms for Closest String, obtaining improvements (in terms of the (constant) bases of exponential functions) over previous deterministic results exploiting small alphabet sizes.

Variants. Nishimura and Simjour [140] presented parameterized enumeration algorithms for Closest String and the slightly more general Neighbor String problem, exploiting the parameters alphabet size $|\Sigma|$ and maximum Hamming distance d. Creignou et al. [63] initiated a study of parameterized enumeration with ordering for Closest String and other problems, proposing a general strategy for this task. Boucher and Omar [35] derived results on the hardness of counting the number of closest strings.

Boucher and Ma [34] and Boucher et al. [37] presented several parameterized tractability and intractability results for the Close to Most Strings problem. This problem generalizes Closest String by relaxing the requirements for the solution: the algorithm may choose to select a given number of "outliers" among input strings, which are then simply ignored. Indeed,

Closest String is the special case where no outliers are allowed. For example, Close to Most Strings is fixed-parameter tractable for the combined parameter maximum Hamming distance d and number k of input strings [37].

Hermelin and Rozenberg [104] introduced the Closest String with Wildcards problem, where the input strings may contain wildcard letters '$*$' that match with every other letter of the alphabet Σ. The solution is required to be without wildcard letters. Clearly, Closest String is the special case where the input strings are without wildcards. Some results for Closest String can be adapted, but new techniques had to be developed for this more general problem to obtain fixed-parameter tractability results. Amir et al. [12] introduced a generalization of Closest String more suitable for clustering applications; here, one has to deal with determining *several* center (closest) strings.

Closest String asks for a solution that has small "radius". Amir et al. [8] introduced the variant where one asks for small radius and small distance sum. Their algorithms work for three-string inputs; the complexity for $k \geq 4$ input strings remains open. Moreover, Lee et al. [118] developed polynomial-time algorithms for computing the (Hamming distance) consensus of three *circular* strings as motivated by biological applications.

Most research focused on the Hamming metric as a distance measure. Several further distance measures such as edit distance, swap distance, reversal distance, or rank distance have been proposed [11, 67, 137]. So far, there are only few parameterized complexity results here. A further variant of Closest String called Shared Center, motivated by applications in haplotype inference in biology, has also been studied [57].

We conclude this subsection with a very unspecific and general challenge based on the following observation. Consensus problems play a prominent role not only in the context of string problems, but also in the context of computational social choice [21, 38]. For example, compare Closest String with the NP-hard Kemeny Rank Aggregation problem. For the latter, given a set of permutations (in other words, every letter appears exactly once in each input string) one seeks a consensus permutation that minimizes the *sum* of inversions (that is, the number of "bubble sort operations") to the input permutations. Bachmaier et al. [15] started an investigation of "maximum rank aggregation problems", in particular including the "maximum version" of Kemeny Rank Aggregation. Among other things they showed how the above-mentioned search tree approach for Closest String [94] can be extended to this setting. Similar parameterized complexity studies as for Closest String have been performed for Kemeny Rank Aggrega-

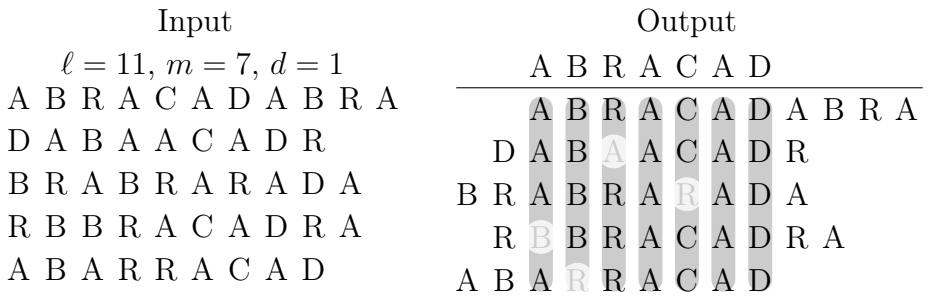

Figure 2: Closest Substring

tion [19, 20, 22], leading to the natural quest for a deeper understanding of interactions and relations between consensus problems from both areas.

Challenge 3. *What are common features (problems, methods, techniques) that are used in deriving parameterized complexity results in computational social choice (particularly, rank aggregation problems) and stringology (particularly, consensus string problems)?*

Interestingly, Aziz et al. [14] closely connected Closest String with minimax approval voting.

3.2 Closest Substring

Closest Substring is the computationally harder sister problem of Closest String. Here, one searches for a consensus string that is close to a fixed-length *substring* in every input string. Figure 2 shows an example.

> Closest Substring
> **Instance:** A set of k maximum-length-ℓ strings $s_1, \ldots, s_k$ over an alphabet Σ, and positive integers d and m.
> **Question:** Is there a length-m string $s \in \Sigma^*$ such that each of $s_1, \ldots, s_k$ has a length-m substring with Hamming distance at most d to s?

Closest Substring can be trivially solved by considering $(\ell - m + 1)^k$ instances of the Closest String problem; clearly, this is inefficient if m is significantly smaller than ℓ and for already moderate values of k. Another straightforward exhaustive search algorithm is to test all possible candidate solution strings, resulting in a running time of $O(|\Sigma|^m \cdot k\ell^2)$. On the negative side, even for constant-size alphabets Closest Substring is W[1]-hard for the parameters maximum Hamming distance d and number k of input

strings [72, 75, 127]. On the positive side, Marx [127] provided algorithms running in $|\Sigma|^{d \log d + 2} \cdot (k \cdot \ell)^{O(\log d)}$ time and running in $|\Sigma|^d \cdot 2^{kd} \cdot d^{O(d \log \log k)} \cdot (k \cdot \ell)^{O(\log \log k)}$ time. Assuming the ETH, these algorithms again are shown to be close to optimality [121, 127]. Moan and Rusu [135] and Ma and Sun [124] investigated variants of Closest Substring where the pairwise distance between input strings is bounded and showed that the (parameterized) hardness results for Closest Substring remain valid.

Challenge 4. *Given the notorious computational hardness of* Closest Substring *and related problems, do there exist parameterizations that allow for fixed-parameter tractability results? In particular, can analysis of real-world input instances lead to useful data-driven parameterizations?*

Variants. Consensus Patterns is the same as Closest Substring except that one does not want to find a substring with small maximum distance but a substring with a small *sum* of distances. Note that while Closest String becomes trivially polynomial-time solvable when moving to sum of distances instead of maximum distance, Consensus Patterns is NP-complete [120]. Compared to Closest Substring, however, in terms of fixed-parameter tractability there are more encouraging results. While for constant-size alphabets Consensus Patterns remains W[1]-hard for the parameter number k of input strings, for the other standard parameters it becomes fixed-parameter tractable as long as the alphabet size is bounded [127].

Distinguishing Substring Selection generalizes Closest Substring by having "good" and "bad" input strings and searching for a solution string that is far away from all substrings of good strings but close to at least one substring in every bad string. The terms "good" and "bad" are motivated by applications concerning the design of genetic markers. Historically, Distinguishing Substring Selection was shown W[1]-hard (for constant alphabet size) for all standard parameters (in particular with respect to the parameter maximum Hamming distance d) before Closest Substring—indeed, the corresponding hardness reductions may be considered somewhat easier [95]. Notably, the special case Distinguishing String Selection in terms of complexity is closer to Closest String than to Distinguishing Substring Selection [152]. Note that Distinguishing String Selection has the two special cases Closest String (here the set of good strings is empty) and Farthest String (here the set of bad strings is empty) [152].

Finally, Basavaraju et al. [16] provided a first systematic study on the

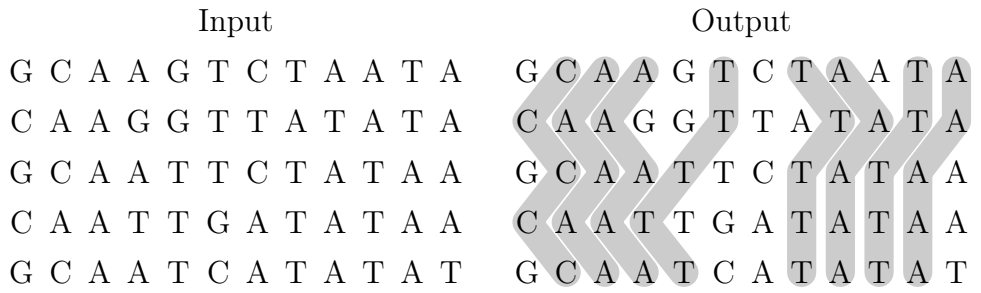

Figure 3: Longest Common Subsequence (example from Skiena [146])

kernelization complexity of many of the problems studied in Section 3. Notably, Hufsky et al. [107] empirically studied polynomial-time data reduction combined with search trees for Closest String. Kernelizability studies, however, are still underrepresented in the context of NP-hard string problems. We thus conclude with the following concrete question, also posed by Basavaraju et al. [16].

Challenge 5. *Does* Closest String *parameterized by the number k of input strings have a size-$k^{O(1)}$ problem kernel?*

4 Common Structure

In this section, we examine the problem of finding a common sub- or superstructure of a given set of strings. The most basic problems are Longest Common Subsequence (Section 4.1), Shortest Common Supersequence (Section 4.3), and Shortest Common Superstring (Section 4.4). In addition, we cover the fairly general Multiple Sequence Alignment problem (Section 4.2), which is of immense importance in biological sequence analysis. The main parameters we consider are the number of strings k, the solution string length m, the maximum input string length ℓ, and the alphabet size $|\Sigma|$.

4.1 Longest Common Subsequence

Longest Common Subsequence is a classic NP-complete problem [84, SR10]. It has applications for example in computational biology, data compression, or file comparison (a variant is used in the Unix diff command) [18]. Figure 3 shows an example.

	alphabet size $\|\Sigma\|$		
parameter	unbounded	parameter	constant
k	W[t]-hard [29]	W[t]-hard [28]	W[1]-hard, $\|\Sigma\| = 2$ [142]
m	W[2]-hard [29]	FPT [E]	FPT [E]
k, m	W[1]-hard [29, 97]	FPT [E]	FPT [E]
ℓ	FPT [E]	FPT [E]	FPT [E]

Table 1: Parameterized complexity of Longest Common Subsequence. Results marked [E] follow from trivial complete enumeration; W[t]-hard refers to any $t \geq 1$.

Longest Common Subsequence
Instance: A set of k maximum-length-ℓ strings $s_1, \ldots, s_k$ over an alphabet Σ and a positive integer m.
Question: Is there a string $s \in \Sigma^*$ of length at least m that is a subsequence of s_i for $i = 1, \ldots, k$?

The case of two strings is well-studied and can be solved in $O(\ell^2)$ time by dynamic programming. An alternative algorithm solves the problem in $O((r + \ell) \log \ell)$ time, where r is the total number of ordered pairs of positions at which the two sequences match [108]. In the worst case, this algorithm has a running time of $O(\ell^2 \log \ell)$; however, in many applications the value of r can be expected to be closer to ℓ (for example in the Unix diff command, where each line occurring in the input is a letter of the alphabet). Thus, this could be considered as a parameterized algorithm for a polynomial-time solvable problem.

For an arbitrary number of strings, the problem is NP-hard even for a binary alphabet [26, 126]. Bodlaender et al. [29] were the first to study the parameterized complexity of Longest Common Subsequence. Currently known results are summarized in Table 1. The problem is W[1]-hard for the possibly most appealing parameter, the number of strings k, even with a binary alphabet [142]. The reduction to prove this claim is a linear parameterized reduction from Clique parameterized by solution size. Hence, assuming ETH a $(k \cdot \ell)^{o(k)}$-time algorithm for Longest Common Subsequence is impossible. Similarly, the original reduction [29] for showing W[2]-hardness of Longest Common Subsequence parameterized by solution length m is a linear parameterized reduction from Dominating Set. This implies that, assuming W[1] $\neq$ FPT, there is no $(k \cdot \ell)^{o(m)}$-time algorithm for Longest Common Subsequence. The reduction, however, produces instances with an unbounded number k of strings. Hence, the following question remains

open.

Challenge 6. *Does* Longest Common Subsequence *admit a* $(k \cdot \ell)^{o(k+m)}$*-time algorithm?*

Note that when the alphabet size $|\Sigma|$ and the length m of the string to be found are parameters, we get a trivial FPT algorithm from enumerating all $|\Sigma|^m$ possible solutions. It would be interesting to see if this parameterization also yields a small kernel.

Challenge 7. *Does* Longest Common Subsequence *have a polynomial-size problem kernel for binary alphabet and parameter* m*, or more generally for the combined parameter* $(m, |\Sigma|)$*?*

A different brute-force algorithm is to enumerate all possible ways in which individual letter positions can be matched exactly over all input strings to generate common subsequences, using a dynamic programming table with $O(\ell^k)$ entries, yielding a running time of $O^*(\ell^k)$ [71]. If we consider as parameter only the maximum input string length ℓ, we can also get a simple brute-force FPT algorithm: For each of the 2^ℓ subsequences of the first string, check whether it is also a subsequence of the other strings, and return the longest common subsequence thus found.

Challenge 8. *Does* Longest Common Subsequence *admit a* $(2-\epsilon)^\ell \cdot k^{O(1)}$*-time algorithm for some* $\epsilon > 0$*?*

Further parameters. Timkovskii [148] shows that Longest Common Subsequence remains NP-hard even when the input strings have length 2 and the maximum number of occurrences occ of a letter over all input strings is 3; several more related results are given. If in addition to occ we use the number of strings k as a parameter, we obtain fixed-parameter tractability [99]: the problem can be reduced to finding a longest path in a directed acyclic graph with $O(\ell \cdot \text{occ}^k)$ vertices.

Blin et al. [26] study Longest Common Subsequence with fixed alphabet size $|\Sigma|$ and unbounded number of strings k, but fixed run-length (that is, maximum number of consecutive identical letters). They show that the problem remains NP-complete even when restricted to strings with run-length at most 1 over an alphabet of size 3 or strings with run-length at most 2 over an alphabet of size 2 (both results are tight).

Extending the approach of Hunt and Szymanski [108], Hsu and Du [105] present an algorithm running in $O(k|\Sigma|(\ell + r))$ time, where r is the number of tuples $(i_1, i_2, \ldots, i_k)$ such that $s_1[i_1] = s_2[i_2] = \cdots = s_k[i_k]$. This clearly

will be most effective for very large alphabets. Irving and Fraser [110] give an algorithm running in $O(k\ell(\ell-m)^{k-1})$ time. This can be seen as a fixed-parameter algorithm for the combined parameter k and number $\ell - m$ of omitted letters.

Challenge 9. *Is* Longest Common Subsequence *fixed-parameter tractable for the parameter number $\ell - m$ of omitted letters?*

Relaxed versions. Motivated by biological applications, several variants of Longest Common Subsequence have been studied where the input strings do not simply consist of letters, but each position is a probability mass function that describes how likely each letter is here (*position weight matrix* in biological literature). In this way, one can talk about the probability of a subsequence. Finding the longest string such that the product of its probability in each of two input strings exceeds some threshold can be done in polynomial time [6]; if however a threshold probability needs to be exceeded in both input strings, the problem becomes NP-hard [6], even for a binary alphabet [65]. The same dichotomy holds for Shortest Common Supersequence [7].

Challenge 10. *Analyze the parameterized complexity of the "most probable subsequence" version of* Longest Common Subsequence.

Guillemot [97] studies the Longest Compatible Sequence problem which can be seen as a variant of Longest Common Subsequence. The input strings are p-sequences, that is, occ $= 1$, and the task is to compute a length-m string s such that for each input string s_i the string s restricted to the alphabet of s_i is a subsequence of s_i. Longest Compatible Sequence is W[1]-hard for the combined parameter (k, ℓ) and fixed-parameter tractable for the parameter $|\Sigma| - m$ (note that $|\Sigma| \geq \ell$) [97].

Constrained versions. A number of variants of Longest Common Subsequence have been examined where the output string needs to have an additional property. Most works consider only two input strings, so we assume this in this paragraph except when noted otherwise.

The Constrained Longest Common Subsequence problem is the generalization where the output must contain each of a given set of f *restriction strings* as subsequence. It has applications in computational biology. The problem can be solved in polynomial time for a single restriction string ($f = 1$) [149], but is NP-hard in general [90]. Chen and Chao [52] give a dynamic programming algorithm with running time $O(\ell^2 \cdot \prod_{i=1}^{f} \rho_i)$, where $\rho_1, \ldots, \rho_f$ are the lengths of the restriction strings; thus, this is a fixed-parameter algorithm for the parameter "total length of the restriction strings t". Bonizzoni et al.

[32] show that the problem is W[1]-hard for the combined parameter $(f, |\Sigma|)$, using a reduction from Shortest Common Supersequence.

The Restricted Longest Common Subsequence problem is the generalization where the output must *not* contain any of a given set of f restriction strings as subsequence. The problem is NP-hard already for two input strings and restriction strings of length two, but can be solved with dynamic programming also for more than two input strings in $O(\ell^{k+f})$ time [91]. A different analysis of this algorithm yields $O(2^t \cdot \ell^k)$ time, where t is the total length of the restriction strings; thus, the problem is fixed-parameter tractable with respect to t. A different dynamic programming algorithm solves the problem with running time $O(\ell^2 \cdot \prod_{i=1}^{f} \rho_i)$, where $\rho_1, \ldots, \rho_f$ are the lengths of the restriction strings [52]; this also implies fixed-parameter tractability with respect to t.

In the Repetition-free Longest Common Subsequence problem [3], each letter must appear at most once in the solution string. The application is to uncover a genome rearrangement where at most one representative of each family of duplicated genes is taken into account. The problem is NP-hard even if occ = 2 [3]. It can be solved in polynomial-time when the number of letters that appear multiple times in the input is a constant [3]. The problem can be solved in randomized $O^*(2^m)$ time and polynomial space [25], that is, it is fixed-parameter tractable with respect to the solution size m. The algorithm uses the multilinear detection technique, an algebraic approach; the idea is to exploit that we can efficiently detect a multilinear monomial of a given degree in an arithmetic circuit, which is a compressed encoding of a multivariate polynomial. On the negative side, the problem does not have a polynomial-size kernel for parameter m unless NP $\subseteq$ coNP/poly [25].

The Doubly-Constrained Longest Common Subsequence [32] generalizes both Constrained Longest Common Subsequence and Repetition-free Longest Common Subsequence by demanding both constraints at the same time. Moreover, the repetition-free constraint is generalized by requiring that the number of occurrences of each letter a in the solution is bounded by some function $\tau(a)$. This models a sequence comparison problem from computational biology. It is NP-complete already with a ternary alphabet but can be solved in time $O^*(m^m 2^{O(m)})$ [32]. This algorithm is based on the color-coding technique, which was introduced by Alon et al. [5] for graph problems. The idea is to color each possible occurrence of a letter in the solution (that is, each pair (σ, i) with $\sigma \in \Sigma, i \in \{1, \ldots, \tau(\sigma)\}$) randomly, and then to look only for solutions that fulfill a certain colorfulness property with respect to this coloring; this restriction makes the task much easier. If we repeat the process frequently enough, we can ensure that the colorfulness property is fulfilled at least once with high probability. By choosing the

Input	Output
G C A A G T C T A A T A	G C A A Δ G T C Δ Δ T A A T A
C A A A G T T A T T A	Δ C A A A G T Δ Δ Δ T A T T A
G C A A G T C C A T A A C	G C A A Δ G T C C A T A A C Δ
G C C A G A C T C A T A	G C C A Δ G A C Δ Δ T C A T A
G C T T C T A A T A	G C Δ Δ Δ T T C Δ Δ T A A T A
	4 0 7 4 4 4 4 4 4 4 0 4 4 4 4 $\sum$: 55

Figure 4: Multiple Sequence Alignment with unit cost function ϕ.

colorings from a perfect hash family, it can also be ensured deterministically that at least one coloring makes the solution colorful. An alternative algorithm based on finite automata has running time $O(m^{f+|\Sigma|} \cdot |\Sigma|\ell^2)$ where f is the number of restriction strings that the solution must contain [73].

Finally, in the Exemplar Longest Common Subsequence problem the alphabet consists of mandatory and optional letters and one is asked to find a longest common subsequence that contains each mandatory letter at least once [31]. On the negative side, it is NP-hard to check whether there is *any* common subsequence (without maximizing its length) even if each mandatory symbol occurs at most three times in each input string. On the positive side, Exemplar Longest Common Subsequence is fixed-parameter tractable for the parameter number of mandatory letters.

4.2 Multiple Sequence Alignment

From the viewpoint of biological applications, the class of multiple sequence alignment problems form arguably the most relevant class of NP-hard string problems. From an alignment of protein, RNA, or DNA sequences, one may infer facts about the evolutionary history of biological species or of the sequences themselves. These problems have as input a set of k strings and the task is to find an alignment of these strings that has minimum cost (see Figure 4 for an example). Herein, an alignment is a rectangular array whose rows correspond to the input strings and may also contain an additional gap symbol Δ. Informally, the goal of any multiple sequence alignment problem is to maximize the total amount of similarity within the alignment columns. There is a variety of possible cost functions to achieve this vaguely defined task. For example, one may only count a column if it does not contain any gap symbol and all its letters are equal. For this scoring function, Multiple

Sequence Alignment is equivalent to Longest Common Subsequence.

In this section, we focus on the so-called sum of pairs score. While it is difficult to give a biological justification for this score, it is relatively easy to work with and has been used in many studies. The sum of pairs score, or rather cost, as the problems are often formulated as minimization problems, is simply the sum of pairwise alignment costs over all pairs of input sequences. The pairwise alignment cost is computed by summing pairwise "mutation" costs over all columns of the alignment. The cost function is a problem-specific symmetric function $\phi : (\Sigma \cup \{\Delta\}) \times (\Sigma \cup \{\Delta\}) \to \mathbb{R}_+$ where $\phi(\sigma, \sigma) = 0$ for each $\sigma \in \Sigma \cup \{\Delta\}$.

Multiple Sequence Alignment (MSA) with SP-Score
Instance: A set S of k maximum-length-ℓ strings $s_1, \ldots, s_k$ over an alphabet Σ, a cost function ϕ and a positive integer m.
Question: Is there an alignment of S that has cost at most m?

Multiple Sequence Alignment is NP-complete for a wide range of cost functions that fulfill the triangle inequality [30, 114]. In particular, Multiple Sequence Alignment is NP-hard for *all* metric cost functions even for binary input strings [70]. This includes the most simple cost function, the *unit cost* function, that assigns a cost of 0 for aligning identical letters and a cost of 1 for aligning a letter with a different letter or with the gap symbol Δ.

Notably, none of the reductions behind these hardness results shows W[1]-hardness for the number of strings k while an $\ell^{O(k)}$-time algorithm can be achieved by standard dynamic programming. Focusing on the algorithmically most fundamental cost function leads to the following challenge.

Challenge 11. *Can* Multiple Sequence Alignment *with unit cost function be solved in* $\ell^{o(k)}$ *time?*

An $\ell^{o(k)}$-time lower bound was presented for the somewhat harder Local Multiple Alignment problem [1].

4.3 Shortest Common Supersequence

Shortest Common Supersequence is another classic NP-hard problem [84, SR10]. Bodlaender et al. [28] mention applications in biology and suggest examining the parameterized complexity for various parameters and problem variants. Figure 5 shows an example.

Shortest Common Supersequence
Instance: A set of k maximum-length-ℓ strings $s_1, \ldots, s_k$ over an alphabet Σ and a positive integer m.
Question: Is there a string $s \in \Sigma^*$ of length at most m that is a supersequence of s_i for $i = 1, \ldots, k$?

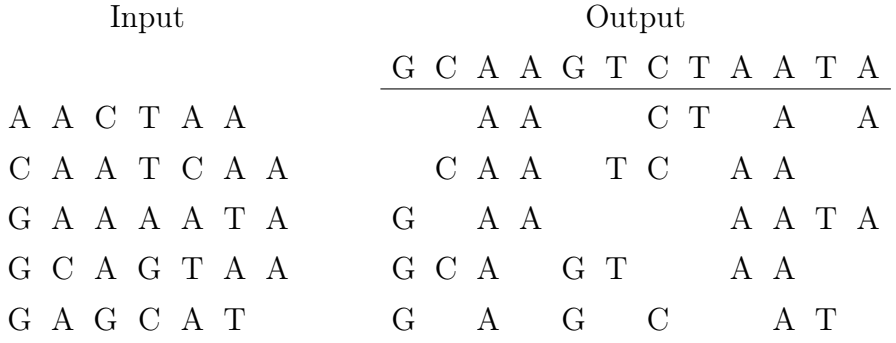

Figure 5: Shortest Common Supersequence

	alphabet size $\|\Sigma\|$					
parameter	unbounded		parameter		constant	
k	W[1]-hard	[102]	W[1]-hard	[102]	W[1]-hard	[142]
m	FPT	[E]	FPT	[E]	FPT	[E]
ℓ	NP-hard $\ell = 2$	[148]	FPT	[E]	FPT	[E]

Table 2: Parameterized complexity of Shortest Common Supersequence. Results marked [E] follow from trivial complete enumeration.

The case of two strings is easily solved in polynomial time by reducing to Longest Common Subsequence. For an arbitrary number of strings, the problem is NP-hard even when all input strings have length two [148], or with a binary alphabet where each string contains exactly two 1's [132].

Known results for the basic parameters are summarized in Table 2. We can trivially enumerate all solutions in $O(|\Sigma|^m)$ time, and with $|\Sigma| \leq m$ and $m \leq \ell \cdot |\Sigma|^\ell$ the other results marked [E] follow. Parameterized by the number of strings k, the problem is W[1]-hard, even when the alphabet size is fixed [142]. As for Longest Common Subsequence, there is no $\ell^{o(k)}$-time algorithm for Shortest Common Supersequence [49].

Further parameters. Shortest Common Supersequence with occ $=$ 1 and parameter $m - \ell$ (number of extra letters) is parameterized equivalent to the Directed Feedback Vertex Set problem [74]. Thus, using the FPT algorithm for the latter [50], we obtain fixed-parameter tractability.

Challenge 12. *Extend the fixed-parameter tractability of* Shortest Common Supersequence *for parameter $m - \ell$ to larger classes of inputs.*

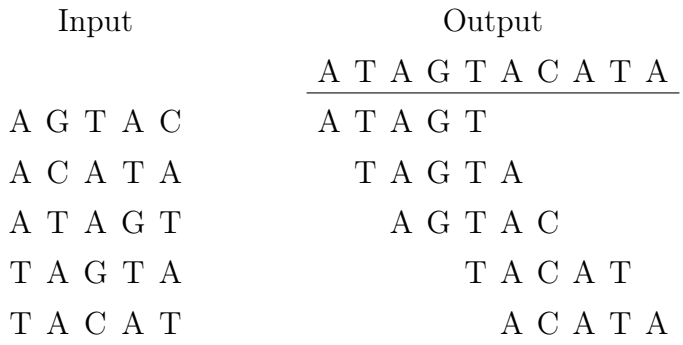

Figure 6: Shortest Common Superstring

Constrained version. Dondi [68] examines a generalization of Shortest Common Supersequence that he calls Constrained Shortest Common Supersequence. The requirement is that in the solution string, each letter a must occur at least $\tau(a)$ times. Only two input strings are considered. Constrained Shortest Common Supersequence is NP-complete even when occ $= 2$ and $\tau(a) \leq 3$ for each $a \in \Sigma$, but is polynomial-time solvable when $\tau(a) \leq 2$ for each $a \in \Sigma$ [68]. Note that in a shortest common supersequence s, letters must be part of the subsequence s_1 or the subsequence s_2 or both (*matching letters*). If we know the matching letters, the solution is easy to construct. Thus, we can solve the problem in $O^*(2^{|s_1|}) = O^*(2^m)$ time by trying all subsets of s_1 that might form the matching letters. By case distinction, this can be improved to $O^*(1.733^m)$ [68].

4.4 Shortest Common Superstring

Shortest Common Superstring is NP-complete [84, SR9]. It has applications in DNA assembly (see e. g. [71]) and data compression (see e. g. [83]). For a survey, see Gevezes and Pitsoulis [85]. Figure 6 shows an example.

> Shortest Common Superstring
> **Instance:** A set S of k length-ℓ strings $s_1, \ldots, s_k$ over an alphabet Σ and a positive integer m.
> **Question:** Is there a string $s \in \Sigma^*$ of length at most m that is a superstring of s_i for $i \in \{1, \ldots, k\}$?

Again, the case of two input strings is polynomial-time solvable, but for an arbitrary number of sequences, the problem is NP-complete even when $|\Sigma| = 2$ or the maximum input string length ℓ is 3 [83].

	alphabet size $\|\Sigma\|$		
parameter	unbounded	parameter	constant
k	FPT [TSP]	FPT [TSP]	FPT [TSP]
m	FPT [E]	FPT [E]	FPT [E]
ℓ	NP-hard $\ell = 3$ [132]	FPT [E]	FPT [E]

Table 3: Parameterized complexity of Shortest Common Superstring. Results marked [TSP] follow from a reduction to Traveling Salesman; results marked [E] follow from trivial complete enumeration.

Bodlaender et al. [28] suggest examining the parameterized complexity of this problem and variants. Evans and Wareham [71] give a survey on parameterized results from the viewpoint of applications in molecular biology, including generalizations based on the applications. The results for the basic parameters are summarized in Table 3. The problem remains NP-complete if the given strings have length 3 and the maximum letter occurrence over all strings is 8 [132], or if all strings are of the form 10^p10^q with $p, q \geq 0$ [133]. Again, we can trivially enumerate all solutions in $O(|\Sigma|^m)$ time, and with $|\Sigma| \leq m$ and $m \leq \ell \cdot |\Sigma|^\ell$ the other results marked [E] follow.

Any superstring can be created by concatenating the input strings in some order and then merging *overlaps*, that is, if there is some string s that is both a suffix of an input string and a prefix of the next input string in the ordering, then we need s only once in the superstring. In a solution superstring, two adjacent strings will always have maximum overlap. The maximum overlap can be calculated in linear time. Thus, after $O(k^2\ell)$ time preprocessing, we can simply try all $k!$ possible orders of the input strings in the common superstring, and solve Shortest Common Superstring in $O(k! + k^2\ell)$ time. Alternatively, we can reduce to the Traveling Salesman problem (TSP) by creating a vertex for each of the k input strings and weighing an arc (s_1, s_2) by the number of nonmatched letters in s_1 in the maximum overlap with s_2 (Figure 7). A minimum-weight path that visits all vertices then corresponds to a shortest common superstring. (Note that we do not seek a round-trip, that is, we can start at any vertex and do not need to return to the start after having visited all other vertices.) For TSP, we can use a classic exponential-space dynamic programming algorithm [17, 103] to solve the problem in $O(k^2 2^k + k^2\ell)$ time. No algorithm for TSP that is faster than $O^*(2^k)$ is known. Note that the edge weights in the TSP instances resulting from this reduction are not symmetric, which unfortunately makes many popular solution approaches for TSP inapplicable. For the special case where

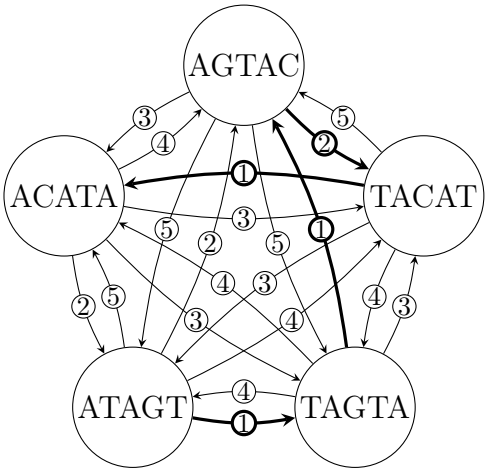

Figure 7: TSP instance for the Shortest Common Superstring instance from Figure 6

the length of the input strings ℓ is three, Shortest Common Superstring can be solved in $O^*(3^{k/3}) = O^*(1.443^k)$ time [88]; more generally, when ℓ is bounded by some constant c, the problem can be solved in randomized $O^*(2^{(1-f(c))\ell})$ time, where $f(c) = 1/(1+2c^2)$ [89].

Challenge 13. *Does* Shortest Common Superstring *admit a* $(2-\epsilon)^k \cdot k^{O(1)}$*-time algorithm for some* $\epsilon > 0$*?*

Note that the challenge does not become any easier when assuming a binary alphabet [150].

Variants. Bonizzoni et al. [33] consider two variations of Shortest Common Superstring, where in addition to the set of strings we are given extra input that restricts possible solutions; the aim is not to cover all strings anymore, but a maximum-size subset. In Swapped Common Superstring, we are additionally given a string t, and the solution string must be a *swap order* of t, that is, it must be obtainable from t by nonoverlapping swaps of adjacent letters. This problem is NP-complete [92], but fixed-parameter tractable with respect to the number of input strings covered in the solution [33]. In Restricted Common Superstring, we are additionally given a multiset $\mathcal{M}$ of letters from the same alphabet, and the solution string must be an ordering of $\mathcal{M}$. This problem is NP-complete, even with binary alphabet or input string length bounded by 2 [60]. For the number of input strings covered in the solution, the problem is W[1]-hard; however, with the additional parameter of the maximum input string length ℓ, it becomes

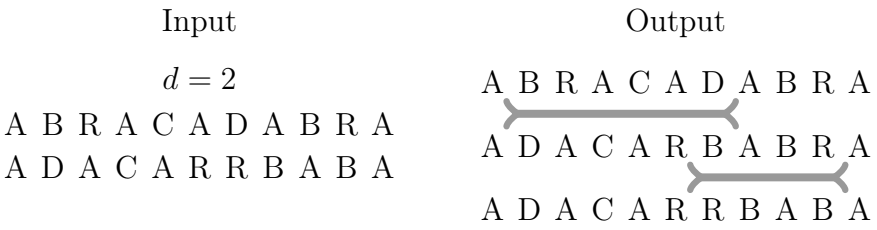

Figure 8: Sorting By Reversals

fixed-parameter tractable [33]. Both fixed-parameter algorithms are based on the color-coding technique. On the negative side, the authors show that the parameterizations that yield fixed-parameter tractability do not admit a polynomial kernel, unless NP $\subseteq$ coNP/poly [33].

5 Distances

The problems in this section aim at answering a common question: how similar are two given strings? Easy answers can be obtained by counting the number of local operations that are necessary to transform one string into the other. The most important examples here are the Hamming distance and the Levenshtein distances. More generally, for any combination of insertions, deletions, single-letter changes, and adjacent swaps, the edit distance can be computed in polynomial time, a notable exception being the edit distance where deletions and adjacent swaps are allowed. Computing this distance is doable in polynomial time for constant-size alphabets [131] but NP-hard in general [151], and fixed-parameter tractable if parameterized by the distance [2]. Some models, however, require nonlocal operations. Usually, this nonlocality makes the distance computation more challenging.

5.1 Reversal and Transposition Distances

A *rearrangement* is a large-scale operation that transforms a string. The study of rearrangements is motivated by the evolution of genomes during which different types of rearrangements occur [79]. One of the most-studied rearrangement operation is the *reversal*, where the order of the letters in a substring is reversed. The reversal distance between two strings is the number of reversals needed to transform one string into the other; see Figure 8 for an example.

String Reversal Distance
Instance: Two strings s_1 and s_2 of length ℓ and an integer d.
Question: Is the reversal distance between s_1 and s_2 at most d?

The problem is nontrivially posed only if each letter occurs with the same frequency in s_1 and s_2; such strings are called *balanced*. For occ $= 1$, String Reversal Distance is equivalent to transforming one permutation into another: the two input strings are balanced and thus $|\Sigma| = \ell$. Since one may assume without loss of generality that $s_2 = 12 \dots \ell$, the problem is called Sorting by Reversals in this case. Sorting by Reversals is NP-hard [47]. String Reversal Distance is NP-hard even if $|\Sigma| = 2$ [59]. Moreover, it remains hard even if $|\Sigma| = 2$ and the run-length (that is, maximum number of subsequent identical letters) is two for one letter and one for the other letter [46]. Besides the alphabet size $|\Sigma|$, the distance d is the most natural parameter. Sorting by Reversals is trivially fixed-parameter tractable for parameter d: If s_1 contains substrings of the form $i(i+1)(i+2)$ or $(i+2)(i+1)i$, then these substrings and their counterparts in s_2 can be replaced by smaller ones. This reduces input instances to equivalent ones of length $O(d)$. This approach, however, does not extend to general strings. Thus, fixed-parameter tractability with respect to d remains open.

Challenge 14. *Is* String Reversal Distance *fixed-parameter tractable for the parameter d?*

Another natural parameter which obviously yields fixed-parameter tractability is the string length ℓ. A trivial search tree algorithm is to branch into all ℓ^2 possibilities for the first reversal and then solve the problem recursively with $d - 1$ for each of the resulting permutations. This gives a running time of $\ell^{O(\ell)}$. Obviously, a significant improvement of this running time is desirable.

Challenge 15. *Does* String Reversal Distance *admit a $2^{O(\ell)}$-time algorithm?*

A further parameter that was proposed for String Reversal Distance is the number b of *blocks*, that is, maximal substrings in which only one letter occurs. This parameter is motivated by the hardness for $|\Sigma| = 2$; for such strings the block number can be much smaller than the string length ℓ. String Reversal Distance can be solved in $O^*((6b)^{2b})$ time [46]. The core idea is to first guess how the blocks get rearranged by the successive reversals: Which block is inside, outside, or split by each reversal? Since the reversal distance between two strings with at most b blocks is $O(b)$, the search tree size depends only on b. Then, the precise end-points of the reversals within each block are computed with a network flow algorithm.

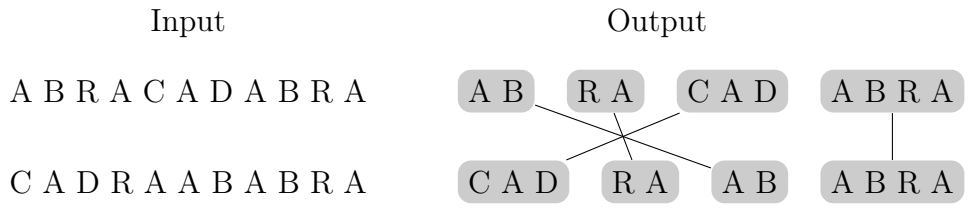

Figure 9: Minimum Common String Partition

Further rearrangement distances. The *transposition* operation is to exchange two consecutive substrings. In the String Transposition Distance problem one asks whether one string can be transformed into another by at most d transpositions. Similarly to the reversal case, String Transposition Distance is NP-hard even if occ $= 1$ [43] or $|\Sigma| = 2$ [144]. A further variant of rearrangement operations are *prefix* reversals and transpositions, where the first letter of the string must be affected by the rearrangement. Motivated by applications in genomics one may also take into account the *orientation* (or *sign*) of the elements of the string when performing a reversal. Notably, the signed version of Sorting By Reversals is solvable in polynomial time. However, for occ $= 2$ [144] the problem becomes NP-hard. Moreover, Signed String Reversal Distance and Signed String Transposition Distance are NP-hard even for unary alphabet, that is, $|\Sigma| = 1$ [46]. The above-mentioned fixed-parameter algorithm for the parameter block number b can be extended (with different running times) to many other rearrangement distances, including transposition distance, and signed and prefix variants of reversal distance and transposition distance.

5.2 Minimum Common String Partition

The Minimum Common String Partition problem aims at splitting one input string into few substrings which can be rearranged to obtain the other substring. Formally, a *common string partition* of two strings s_1 and s_2 is a partition $\mathcal{P}$ of s_1 into $s_1 = s_1^1 \cdot s_1^2 \cdot \ldots \cdot s_1^{d-1} \cdot s_1^d$ and of s_2 into $s_2 = s_2^1 \cdot s_2^2 \cdot \ldots \cdot s_2^{d-1} \cdot s_2^d$ such that there exists a permutation M of $\{1, \ldots, d\}$ where each s_1^i is the same string as $s_2^{M(i)}$; see Figure 9. Here, d represents the *size* of the partition, and the substrings s_i^j are called *blocks*. The problem, introduced independently by Chen et al. [51], Goldstein et al. [87], and Swenson et al. [147] (who call it Sequence Cover) is defined as follows.

Minimum Common String Partition
Instance: Two strings s_1 and s_2 of length ℓ and an integer d.
Question: Is there a common string partition of s_1, s_2 of size at most d?

Similar to reversal distance, two strings have a common string partition only if they are balanced, that is, each letter appears with the same frequency in both strings.

The problem can be seen as a relaxation of problems like Sorting By Transpositions, where one aims only at identifying conserved regions without building a precise evolution scenario. In particular, the number of blocks is a good approximation of the actual transposition distance. Minimum Common String Partition can also be seen as a way of creating a bijection between elements of each string. This can be used to identify similar genes across different genomes [51].

Minimum Common String Partition is NP-hard even if $|\Sigma| = 2$ or if occ $= 2$ [87]. Damaschke [66] identified Minimum Common String Partition as a challenging problem for parameterized algorithmics. He described a fixed-parameter algorithm for the combined parameter block number d and *repetition number* r, defined as the maximum power of any substring of s_1 or s_2. Herein, the *power* of a string w is a number r such that there is a string u with $w = u^r$. Minimum Common String Partition can also be solved in $O((2x)^d d!\ell)$ time where x is the maximum difference between a block size and the average block size ℓ/d [112]. A further fixed-parameter algorithm has running time $O^*((\text{occ})!^d)$. This running time was subsequently improved to $O^*(\text{occ}^{2d})$ [44]. The main idea behind the improved algorithm is as follows. Assume that some elements of both strings, called *seeds*, are already matched across the two strings. Draw a graph over the set of elements of both strings as follows. Add an edge between any pair of elements (one in each string) which may be matched if they are in the same block as a seed. If the resulting graph admits a perfect matching, then there exists a common string partition with as many blocks as seeds. Otherwise, some connected component does not have a perfect matching. A new seed can be found using one of the elements of this component and an element with the same letter in the other sequence. The overall number of options for this new seed is occ^2. The running time bound follows from the fact that at most d seeds need to be considered which bounds the depth of the search tree.

Finally, Minimum Common String Partition is fixed-parameter tractable for the parameter d [41]. The corresponding algorithm, however, has an impractical running time of $O^*(d^{21d^2})$. This algorithm uses the following framework, also proposed by Damaschke [66]. First split the input strings into $O(d)$ pieces. Then guess which pieces are completely contained in a

block. Continue recursively on the remaining pieces, until all blocks have been discovered. The main technical difficulty is to reduce the size of the remaining pieces in order to find at least one new block in each splitting round.

Variants. A *signed* variant where each element is given a sign (+ or −) and a block of s_1 may be matched either to an identical block in s_2, or to its reverse (where both the order and the signs of the elements are inverted) has also been considered [51]. To deal with unbalanced strings, the following model has been proposed [44]: some elements may be deleted from each input string, but only between two consecutive blocks and only as few as necessary so that the resulting strings are balanced (that is, the same letter may not be deleted from both strings). An efficient algorithm that solves both of these extensions would be desirable. Towards this goal, one could first address the following problem.

Challenge 16. *Is* Signed Minimum Common String Partition *fixed-parameter tractable for the parameter d?*

A generalization of Minimum Common String Partition, where blocks are allowed to have a small number of mismatches, and may additionally not partition exactly the input strings is studied by Lopresti and Tomkins [123] under the name Block edit distance. Most variants of Block edit distance are NP-hard; some interesting special cases can be solved in polynomial-time. Gu et al. [96] consider the one-sided Minimum Common String Partition problem, termed Exact Block Cover: Here one sequence is already partitioned into d blocks and the task is to partition the other sequence accordingly. Exact Block Cover is NP-complete even with binary alphabet, but polynomial-time solvable when occ ≤ 3. Further, it can be solved in $O^*(2^d)$ time.

5.3 Other Distances

The following string distances are particularly complex. They have been the subject of very little or no studies in terms of fixed-parameter tractability. In the first of these distances, the task is to find common subsequences of input strings which are permutations that are close with respect to some distance measure on permutations.

Exemplar δ Distance (where δ is a given distance function over permutations)
Instance: Two strings s_1 and s_2 over an alphabet Σ and an integer d.
Question: Are there p-sequences s'_1, s'_2 of length $|\Sigma|$ such that s'_1 is a subsequence of s_1, s'_s is a subsequence of s_2, and $\delta(s'_1, s'_2) \leq d$?

For each distance function δ, we obtain a different problem. When $d = 0$, these problems coincide, thus leading to the 0-Exemplar Distance problem, which has a straightforward formulation.

0-Exemplar Distance
Instance: Two strings s_1 and s_2 over an alphabet Σ.
Question: Is there a p-sequence s of length $|\Sigma|$ which is a common subsequence of s_1 and s_2?

0-Exemplar Distance is NP-hard [113] even if occ = 2. This implies NP-hardness of Exemplar δ Distance for all distance functions δ. Furthermore, for many distance functions, including Hamming distance and breakpoint distance, NP-hardness can be shown even if one of the two input strings is a permutation [13, 40].

The last problem we consider, Maximal Strip Recovery, aims at grouping elements of each string into nonoverlapping *strips* [153]. Here, a strip is a common subsequence of length at least 2; see Figure 10. In the proposed application, "single" elements which cannot be attached to any strip are considered as noise which can be deleted. The number d of such elements gives a measure of dissimilarity between the two input strings.

Maximal Strip Recovery
Instance: Two strings s_1, s_2 of length ℓ and an integer d.
Question: Are there q strings $(w_i)_{1 \leq i \leq q}$, each of length at least 2, and a permutation σ of $\{1, \ldots, q\}$ such that: $w_1 \cdot \ldots \cdot w_q$ is a subsequence of s_1, $w_{\sigma(1)} \cdot \ldots \cdot w_{\sigma(q)}$ is a subsequence of s_2, and $\sum_{i=1}^{q} |w_i| \geq \ell - d$?

Maximal Strip Recovery is NP-hard, even if occ = 1 [53] or if we force the strips w_i to actually be substrings instead of subsequences [45]. The problem restricted to permutations is fixed-parameter tractable for d [42, 111], the current best running time being $O^*(2.36^d)$ [42]. It is unclear whether this result extends to strings.

Challenge 17. *Is* Maximal Strip Recovery *fixed-parameter tractable for the parameter d?*

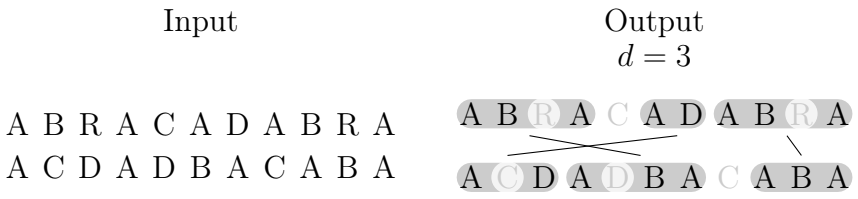

Figure 10: Maximum Strip Recovery

6 Miscellaneous

In this section, we point to some further NP-hard string problems which do not fit into the above classification but nevertheless yield interesting research questions.

String problems with variables. The NP-hard String Morphism problem is to generate from a source string s_1 over alphabet Σ_1 a target string s_2 over an alphabet Σ_2 by uniformly replacing letters in Σ_1, called variables, by strings from Σ_2^*. The task is to decide whether such a replacement exists. String Morphism is NP-hard even for very restricted inputs [77]. Fernau et al. [78] consider different parameters such as $|\Sigma_1|$, $|\Sigma_2|$, the maximum length ω of the strings substituted for the variables in Σ_1, and the maximum number occ_1 of occurrences of a letter in s_1. For a wide range of combined parameters, for example for the combined parameter $(|\Sigma_1|, |\Sigma_2|, \text{occ}_1)$, the problem becomes W[1]-hard; a fixed-parameter algorithm exists for example for the combined parameter $(|\Sigma_1|, \omega)$ [78]. String Morphism is a special case of the problem of deciding whether a word equation [143] is solvable; further investigations could thus address this more general problem.

Collision-aware string partitioning. This new family of string problems is motivated by applications in biotechnology [62]. Informally, these problems are defined as follows: partition a string into substrings such that no two substrings of the partition are similar. For example, one may demand that all substrings of the partition are unequal or that no substring of the partition is a prefix of another substring. All of the considered problems remain hard even for binary strings [61, 62]. The parameterized complexity of the problems is open.

Local search for hard string problems. A common heuristic for hard optimization problems is local search. This approach works as follows. Each problem is equipped with a set of feasible solutions and each solution has

an objective value. Start with some some solution. Then, check whether there is a better solution that is in a suitably defined neighborhood of the current solution. If yes, then continue the process with this solution. Otherwise, output the current, locally optimal solution. For the four string problems Closest String, Longest Common Subsequence, Shortest Common Supersequence, and Shortest Common Superstring the set of feasible solutions are strings on the input alphabet Σ. One possible neighborhood of a string s is the set of strings with Hamming distance at most d. This neighborhood has size $|s|^{O(d)}$. Thus, an interesting question is whether this neighborhood can be efficiently searched, for example in $f(d) \cdot |s|^{O(1)}$ time. For all four problems it is W[1]-hard to decide for a given solution string whether there is a better solution string within Hamming distance d [101]. Moreover, if the ETH is true, then for all problems except Shortest Common Superstring it is impossible to find an algorithm with running time $\ell^{o(d)}$ [101]. Despite this initial set of negative results, local search should still be a worthwhile research direction in the realm of string problems. The following challenge for Shortest Common Superstring demonstrates how diverse the questions in this area can be. Recall that Shortest Common Superstring may be reduced to finding an optimal tour in a TSP instance. Thus, the set of feasible solutions can be also seen as a permutation of the set of input strings. Now the neighborhood of a solution is defined by a suitable distance between permutations, for example swap distance which counts the number of pairwise exchanges of (not necessarily adjacent) elements needed to transform one permutation into the other.

Challenge 18. *Is the following problem fixed-parameter tractable with respect to d? Given a set of k strings $\{s_1, \ldots, s_k\}$ and a permutation π of $\{s_1, \ldots, s_k\}$ such that the superstring corresponding to π has length m, is there a permutation π' of $\{s_1, \ldots, s_k\}$ such that the superstring corresponding to π' has length $m' < m$ and the swap distance between π and π' is at most d?*

To answer the challenge it might be useful to exploit known results on the parameterized complexity of local search variants of TSP [100, 129].

7 Outlook

NP-hard string problems offer a rich working area for multivariate algorithmics research. In particular, compared to graph-theoretic problems there are several issues that so far have been widely neglected:

- Kernelization issues [98, 117] including topics such as Turing kernelization [117] or partial kernelization [20].

- Parameter hierarchies [116] for gaining an even more refined view of parameterized complexity. To identify new nontrivial parameters and parameter relationships, one might draw from the rich set of results on combinatorics on words [64].

- Algorithm engineering and empirical validation [106] of fixed-parameter string algorithms.

- Parameterized approximation algorithms [128] for string problems.

- Distance to triviality [99] and width-based parameterizations (such as treewidth) are very successful in algorithmic graph theory—are there analogous types of parameterizations for string problems? A first step in this direction was undertaken by Reidenbach and Schmid [145] who study a width-based parameterization for the NP-complete membership problem for pattern languages.

Finally, we clearly did not cover all relevant research on multivariate algorithmics for (unweighted) string problems. In particular, certain types of "annotated" and more general problems such as arc-annotated string problems [4, 24, 27] as motivated by applications in analyzing RNA sequences have been completely omitted.

Acknowledgment. We thank Henning Fernau (Universität Trier) and Stéphane Vialette (Université Paris-Est Marne-la-Vallée) for reading a previous draft of the manuscript and providing us with their constructive feedback.

References

[1] A. Abboud, V. V. Williams, and O. Weimann. Consequences of faster alignment of sequences. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP '14)*, volume 8572 of *LNCS*, pages 39–51. Springer, 2014.

[2] F. N. Abu-Khzam, H. Fernau, M. A. Langston, S. Lee-Cultura, and U. Stege. Charge and reduce: A fixed-parameter algorithm for string-to-string correction. *Discrete Optimization*, 8(1):41–49, 2011.

[3] S. S. Adi, M. D. V. Braga, C. G. Fernandes, C. E. Ferreira, F. V. Martinez, M.-F. Sagot, M. A. Stefanes, C. Tjandraatmadja, and Y. Wakabayashi. Repetition-free longest common subsequence. *Discrete Applied Mathematics*, 158(12):1315–1324, 2010.

[4] J. Alber, J. Gramm, J. Guo, and R. Niedermeier. Computing the similarity of two sequences with nested arc annotations. *Theoretical Computer Science*, 312(2-3):337–358, 2004.

[5] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[6] A. Amir, Z. Gotthilf, and B. R. Shalom. Weighted LCS. *Journal of Discrete Algorithms*, 8(3):273–281, 2010.

[7] A. Amir, Z. Gotthilf, and B. R. Shalom. Weighted shortest common supersequence. In *Proceedings of the 18th International Symposium on String Processing and Information Retrieval (SPIRE '11)*, volume 7024 of *LNCS*, pages 44–54. Springer, 2011.

[8] A. Amir, G. M. Landau, J. C. Na, H. Park, K. Park, and J. S. Sim. Efficient algorithms for consensus string problems minimizing both distance sum and radius. *Theoretical Computer Science*, 412(39):5239–5246, 2011.

[9] A. Amir, H. Paryenty, and L. Roditty. Approximations and partial solutions for the consensus sequence problem. In *Proceedings of the 18th International Symposium on String Processing and Information Retrieval (SPIRE '11)*, volume 7024 of *LNCS*, pages 168–173. Springer, 2011.

[10] A. Amir, H. Paryenty, and L. Roditty. Configurations and minority in the string consensus problem. In *Proceedings of the 19th International Symposium on String Processing and Information Retrieval (SPIRE '12)*, volume 7608 of *LNCS*, pages 42–53. Springer, 2012.

[11] A. Amir, H. Paryenty, and L. Roditty. On the hardness of the consensus string problem. *Information Processing Letters*, 113(10-11):371–374, 2013.

[12] A. Amir, J. Ficler, L. Roditty, and O. S. Shalom. On the efficiency of the Hamming c-centerstring problems. In *Proceedings of the 25th Annual Symposium on Combinatorial Pattern Matching (CPM '14))*, volume 8486 of *LNCS*, pages 1–10. Springer, 2014.

[13] S. Angibaud, G. Fertin, I. Rusu, A. Thévenin, and S. Vialette. On the approximability of comparing genomes with duplicates. *Journal of Graph Algorithms and Applications*, 13(1):19–53, 2009.

[14] H. Aziz, S. Gaspers, J. Gudmundsson, S. Mackenzie, N. Mattei, and T. Walsh. Computational aspects of multi-winner approval voting. In *Proceedings of the 8th Multidisciplinary Workshop on Advances in Preference Handling*. AAAI Press, 2014.

[15] C. Bachmaier, F.-J. Brandenburg, A. Gleißner, and A. Hofmeier. On maximum rank aggregation problems. In *Proceedings of the 24th International Workshop on Combinatorial Algorithms (IWOCA '13)*, volume 8288 of *LNCS*, pages 14–27. Springer, 2013.

[16] M. Basavaraju, F. Panolan, A. Rai, M. S. Ramanujan, and S. Saurabh. On the kernelization complexity of string problems. In *20th International Conference on Computing and Combinatorics (COCOON '14)*, volume 8591 of *LNCS*, pages 141–153. Springer, 2014.

[17] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, 1962.

[18] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *Proceedings of the 7th International Symposium on String Processing and Information Retrieval (SPIRE '00)*, pages 39–48. IEEE, 2000.

[19] N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier, and F. A. Rosamond. Fixed-parameter algorithms for Kemeny rankings. *Theoretical Computer Science*, 410(45):4554–4570, 2009.

[20] N. Betzler, J. Guo, C. Komusiewicz, and R. Niedermeier. Average parameterization and partial kernelization for computing medians. *Journal of Computer and System Sciences*, 77(4):774–789, 2011.

[21] N. Betzler, R. Bredereck, J. Chen, and R. Niedermeier. Studies in computational aspects of voting—a parameterized complexity perspective. In *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *LNCS*, pages 318–363. Springer, 2012.

[22] N. Betzler, R. Bredereck, and R. Niedermeier. Theoretical and empirical evaluation of data reduction for exact Kemeny rank aggregation. *Autonomous Agents and Multi-Agent Systems*, 28(5):721–748, 2014.

[23] R. van Bevern, M. Mnich, R. Niedermeier, and M. Weller. Interval scheduling and colorful independent sets. *Journal of Scheduling*, 2014. Available online.

[24] G. Blin, M. Crochemore, and S. Vialette. Algorithmic aspects of arc-annotated sequences. In *Algorithms in Molecular Biology: Techniques, Approaches, and Applications*. Wiley, 2011.

[25] G. Blin, P. Bonizzoni, R. Dondi, and F. Sikora. On the parameterized complexity of the repetition free longest common subsequence problem. *Information Processing Letters*, 112(7):272–276, 2012.

[26] G. Blin, L. Bulteau, M. Jiang, P. J. Tejada, and S. Vialette. Hardness of longest common subsequence for sequences with bounded run-lengths. In *Proceedings of the 23rd Annual Symposium on Combinatorial Pattern Matching (CPM '12)*, volume 7354 of *LNCS*, pages 138–148. Springer, 2012.

[27] G. Blin, M. Jiang, and S. Vialette. The longest common subsequence problem with crossing-free arc-annotated sequences. In *Proceedings of the 19th International Symposium on String Processing and Information Retrieval (SPIRE '12)*, volume 7608 of *LNCS*, pages 130–142. Springer, 2012.

[28] H. L. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett, and H. T. Wareham. Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences*, 11(1):49–57, 1995.

[29] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and H. T. Wareham. The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science*, 147(1&2):31–54, 1995.

[30] P. Bonizzoni and G. D. Vedova. The complexity of multiple sequence alignment with SP-score that is a metric. *Theoretical Computer Science*, 259(1):63–79, 2001.

[31] P. Bonizzoni, G. D. Vedova, R. Dondi, G. Fertin, R. Rizzi, and S. Vialette. Exemplar longest common subsequence. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):535–543, 2007.

[32] P. Bonizzoni, G. D. Vedova, R. Dondi, and Y. Pirola. Variants of constrained longest common subsequence. *Information Processing Letters*, 110(20):877–881, 2010.

[33] P. Bonizzoni, R. Dondi, G. Mauri, and I. Zoppis. Restricted and swap common superstring: A multivariate algorithmic perspective. *Algorithmica*, 2014. Available online.

[34] C. Boucher and B. Ma. Closest string with outliers. *BMC Bioinformatics*, 12(S-1):S55, 2011.

[35] C. Boucher and M. Omar. On the hardness of counting and sampling center strings. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(6):1843–1846, 2012.

[36] C. Boucher, D. G. Brown, and S. Durocher. On the structure of small motif recognition instances. In *Proceedings of the 15th International Symposium on String Processing and Information Retrieval (SPIRE '08)*, volume 5280 of *LNCS*, pages 269–281. Springer, 2008.

[37] C. Boucher, G. M. Landau, A. Levy, D. Pritchard, and O. Weimann. On approximating string selection problems with outliers. *Theoretical Computer Science*, 498:107–114, 2013.

[38] F. Brandt, V. Conitzer, and U. Endriss. Computational social choice. In *Multiagent Systems*, pages 213–283. MIT Press, 2013.

[39] R. Bredereck, J. Chen, P. Faliszewski, J. Guo, R. Niedermeier, and G. J. Woeginger. Parameterized algorithmics for computational social choice: nine research challenges. *Tsinghua Science and Technology*, 19 (4):358–373, 2014.

[40] L. Bulteau and M. Jiang. Inapproximability of (1, 2)-exemplar distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(6):1384–1390, 2013.

[41] L. Bulteau and C. Komusiewicz. Minimum common string partition parameterized by partition size is fixed-parameter tractable. In *Proceedings the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA '14)*, pages 102–121. SIAM, 2014.

[42] L. Bulteau, G. Fertin, M. Jiang, and I. Rusu. Tractability and approximability of maximal strip recovery. *Theoretical Computer Science*, 440-441:14–28, 2012.

[43] L. Bulteau, G. Fertin, and I. Rusu. Sorting by transpositions is difficult. *SIAM Journal on Discrete Mathematics*, 26(3):1148–1180, 2012.

[44] L. Bulteau, G. Fertin, C. Komusiewicz, and I. Rusu. A fixed-parameter algorithm for minimum common string partition with few duplications. In *Proceedings of the 13th International Workshop on Algorithms in Bioinformatics (WABI '13)*, pages 244–258, 2013.

[45] L. Bulteau, G. Fertin, and I. Rusu. Maximal strip recovery problem with gaps: Hardness and approximation algorithms. *Journal of Discrete Algorithms*, 19(0):1 – 22, 2013.

[46] L. Bulteau, G. Fertin, and C. Komusiewicz. Reversal distances for strings with few blocks or small alphabets. In *Proceedings of the 25th Annual Symposium on Combinatorial Pattern Matching (CPM '14)*, volume 8486 of *LNCS*, pages 50–59. Springer, 2014.

[47] A. Caprara. Sorting by reversals is difficult. In *Proceedings of the 1st Annual International Conference on Research in Computational Molecular Biology (RECOMB '97)*, pages 75–83. ACM, 1997.

[48] J. Chen, B. Chor, M. Fellows, X. Huang, D. W. Juedes, I. A. Kanj, and G. Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005.

[49] J. Chen, X. Huang, I. A. Kanj, and G. Xia. On the computational hardness based on linear FPT-reductions. *Journal of Combinatorial Optimization*, 11(2):231–247, 2006.

[50] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55(5), 2008.

[51] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(4):302–315, 2005.

[52] Y.-C. Chen and K.-M. Chao. On the generalized constrained longest common subsequence problems. *Journal of Combinatorial Optimization*, 21(3):383–392, 2011.

[53] Z. Chen, B. Fu, M. Jiang, and B. Zhu. On recovering syntenic blocks from comparative maps. *Journal of Combinatorial Optimization*, 18(3): 307–318, 2009.

[54] Z.-Z. Chen and L. Wang. Fast exact algorithms for the closest string and substring problems with application to the planted (l, d)-motif model. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(5):1400–1410, 2011.

[55] Z.-Z. Chen, B. Ma, and L. Wang. A three-string approach to the closest string problem. *Journal of Computer and System Sciences*, 78 (1):164–178, 2012.

[56] Z.-Z. Chen, B. Ma, and L. Wang. Randomized and parameterized algorithms for the closest string problem. In *Proceedings of the 25th Annual Symposium on Combinatorial Pattern Matching (CPM '14)*, volume 8486 of *LNCS*, pages 100–109. Springer, 2014.

[57] Z.-Z. Chen, W. Ma, and L. Wang. The parameterized complexity of the shared center problem. *Algorithmica*, 69(2):269–293, 2014.

[58] M. Chimani, M. Woste, and S. Böcker. A closer look at the closest string and closest substring problem. In *Proceedings of the 30th Workshop on Algorithm Engineering and Experiments (ALENEX '11)*, pages 13–24. SIAM, 2011.

[59] D. A. Christie and R. W. Irving. Sorting strings by reversals and by transpositions. *SIAM Journal on Discrete Mathematics*, 14(2):193–206, 2001.

[60] R. Clifford, Z. Gotthilf, M. Lewenstein, and A. Popa. Restricted common superstring and restricted common supersequence. In *Proceedings of the 22nd Annual Symposium on Combinatorial Pattern Matching (CPM '11)*, volume 6661 of *LNCS*, pages 467–478. Springer, 2011.

[61] A. Condon, J. Maňuch, and C. Thachuk. Complexity of a collision-aware string partition problem and its relation to oligo design for gene synthesis. In *Proceedings of the 14th Annual International Conference on Computing and Combinatorics (COCOON '08)*, volume 5092 of *LNCS*, pages 265–275. Springer, 2008.

[62] A. Condon, J. Maňuch, and C. Thachuk. The complexity of string partitioning. In *Proceedings of the 23rd Annual Symposium on Combinatorial Pattern Matching (CPM '12)*, volume 7354 of *LNCS*, pages 159–172. Springer, 2012.

[63] N. Creignou, R. Ktari, A. Meier, J.-S. Müller, F. Olive, and H. Vollmer. Parameterized enumeration with ordering. *CoRR*, abs/1309.5009, 2013.

[64] M. Crochemore and W. Rytter. *Jewels of Stringology.* World Scientific, 2002.

[65] M. Cygan, M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń. Polynomial-time approximation algorithms for weighted LCS problem. In *Proceedings of the 22nd Annual Symposium on Combinatorial Pattern Matching (CPM '11)*, volume 6661 of *LNCS*, pages 455–466. Springer, 2011.

[66] P. Damaschke. Minimum common string partition parameterized. In *Proceedings of the 8th International Workshop on Algorithms in Bioinformatics (WABI '08)*, volume 5251 of *LNCS*, pages 87–98, 2008.

[67] L. P. Dinu and A. Popa. On the closest string via rank distance. In *Proceedings of the 23rd Annual Symposium on Combinatorial Pattern Matching (CPM '12)*, volume 7354 of *LNCS*, pages 413–426. Springer, 2012.

[68] R. Dondi. The constrained shortest common supersequence problem. *Journal of Discrete Algorithms*, 21:11–17, 2013.

[69] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[70] I. Elias. Settling the intractability of multiple alignment. *Journal of Computational Biology*, 13(7):1323–1339, 2006.

[71] P. A. Evans and T. Wareham. Efficient restricted-case algorithms for problems in computational biology. In *Algorithms in Computational Molecular Biology: Techniques, Approaches and Applications*, Wiley Series in Bioinformatics, pages 27–49. Wiley, 2011.

[72] P. A. Evans, A. D. Smith, and H. T. Wareham. On the complexity of finding common approximate substrings. *Theoretical Computer Science*, 306(1-3):407–430, 2003.

[73] E. Farhana and M. S. Rahman. Doubly-constrained LCS and hybrid-constrained LCS problems revisited. *Information Processing Letters*, 112(13):562–565, 2012.

[74] M. R. Fellows, M. T. Hallett, and U. Stege. Analogs & duals of the MAST problem for sequences & trees. *Journal of Algorithms*, 49(1): 192–216, 2003.

[75] M. R. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of motif search problems. *Combinatorica*, 26(2):141–167, 2006.

[76] M. R. Fellows, B. M. P. Jansen, and F. A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3): 541–566, 2013.

[77] H. Fernau and M. L. Schmid. Pattern matching with variables: A multivariate complexity analysis. In *Proceedings of the 24th Annual Symposium on Combinatorial Pattern Matching (CPM '13)*, volume 7922 of *LNCS*, pages 83–94. Springer, 2013.

[78] H. Fernau, M. L. Schmid, and Y. Villanger. On the parameterised complexity of string morphism problems. In *Proceedings of the 33rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '13)*, volume 24 of *LIPIcs*, pages 55–66. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.

[79] G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements*. Computational Molecular Biology. MIT Press, 2009.

[80] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer Verlag, 2006.

[81] M. Frances and A. Litman. On covering problems of codes. *Theory of Computing Systems*, 30(2):113–119, 1997.

[82] A. Frank and É. Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1): 49–65, 1987.

[83] J. Gallant, D. Maier, and J. A. Storer. On finding minimal length superstrings. *Journal of Computer and System Sciences*, 20(1):50–58, 1980.

[84] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[85] T. P. Gevezes and L. S. Pitsoulis. The shortest superstring problem. In *Optimization in Science and Engineering*, pages 189–227. Springer, 2014.

[86] P. Giannopoulos, C. Knauer, and S. Whitesides. Parameterized complexity of geometric problems. *The Computer Journal*, 51(3):372–384, 2008.

[87] A. Goldstein, P. Kolman, and J. Zheng. Minimum common string partition problem: Hardness and approximations. *Electronic Journal of Combinatorics*, 12, 2005.

[88] A. Golovnev, A. S. Kulikov, and I. Mihajlin. Solving 3-superstring in $3^{n/3}$ time. In *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS '13)*, volume 8087 of *LNCS*, pages 480–491. Springer, 2013.

[89] A. Golovnev, A. S. Kulikov, and I. Mihajlin. Solving SCS for bounded length strings in fewer than 2^n steps. *Information Processing Letters*, 114(8):421–425, 2014.

[90] Z. Gotthilf, D. Hermelin, and M. Lewenstein. Constrained LCS: Hardness and approximation. In *Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM '08)*, volume 5029 of *LNCS*, pages 255–262. Springer, 2008.

[91] Z. Gotthilf, D. Hermelin, G. M. Landau, and M. Lewenstein. Restricted LCS. In *Proceedings of the 17th International Symposium on String Processing and Information Retrieval (SPIRE '10)*, volume 6393 of *LNCS*, pages 250–257. Springer, 2010.

[92] Z. Gotthilf, M. Lewenstein, and A. Popa. On shortest common superstring and swap permutations. In *Proceedings of the 17th International Symposium on String Processing and Information Retrieval (SPIRE '10)*, volume 6393 of *LNCS*, pages 270–278. Springer, 2010.

[93] J. Gramm, R. Niedermeier, and P. Rossmanith. Exact solutions for closest string and related problems. In *Proceedings of the 12th International Symposium on Algorithms and Computation (ISAAC '01)*, volume 2223 of *LNCS*, pages 441–453. Springer, 2001.

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

[95] J. Gramm, J. Guo, and R. Niedermeier. Parameterized intractability of distinguishing substring selection. *Theory of Computing Systems*, 39 (4):545–560, 2006.

[96] Q. Gu, P. Hell, and B. Yang. On the exact block cover problem. In *Proceedings of the 10th International Conference on Algorithmic Aspects*

in Information and Management (AAIM '14), volume 8546 of *LNCS*, pages 13–22. Springer, 2014.

[97] S. Guillemot. Parameterized complexity and approximability of the longest compatible sequence problem. *Discrete Optimization*, 8(1): 50–60, 2011.

[98] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.

[99] J. Guo, F. Hüffner, and R. Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC '04)*, volume 3162 of *LNCS*, pages 162–173. Springer, 2004.

[100] J. Guo, S. Hartung, R. Niedermeier, and O. Suchý. The parameterized complexity of local search for TSP, more refined. *Algorithmica*, 67(1): 89–110, 2013.

[101] J. Guo, D. Hermelin, and C. Komusiewicz. Local search for string problems: Brute force is essentially optimal. *Theoretical Computer Science*, 525:30–41, 2014.

[102] M. T. Hallett. *An Integrated Complexity Analysis of Problems from Computational Biology*. PhD thesis, University of Victoria, Canada, 1998.

[103] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[104] D. Hermelin and L. Rozenberg. Parameterized complexity analysis for the closest string with wildcards problem. In *Proceedings of the 25th Annual Symposium on Combinatorial Pattern Matching (CPM '14)*, volume 8486 of *LNCS*, pages 140–149. Springer, 2014.

[105] W. J. Hsu and M. W. Du. Computing a longest common subsequence for a set of strings. *BIT Numerical Mathematics*, 24(1):45–59, 1984.

[106] F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *The Computer Journal*, 51(1):7–25, 2008.

[107] F. Hufsky, L. Kuchenbecker, K. Jahn, J. Stoye, and S. Böcker. Swiftly computing center strings. *BMC Bioinformatics*, 12:106, 2011.

[108] J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5): 350–353, 1977.

[109] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63 (4):512–530, 2001.

[110] R. W. Irving and C. B. Fraser. Two algorithms for the longest common subsequence of three (or more) strings. In *Proceedings of the 3rd Annual Symposium on Combinatorial Pattern Matching (CPM '92)*, volume 644 of *LNCS*, pages 214–229. Springer, 1992.

[111] H. Jiang, Z. Li, G. Lin, L. Wang, and B. Zhu. Exact and approximation algorithms for the complementary maximal strip recovery problem. *Journal of Combinatorial Optimization*, 23(4):493–506, 2012.

[112] H. Jiang, B. Zhu, D. Zhu, and H. Zhu. Minimum common string partition revisited. *Journal of Combinatorial Optimization*, 23(4):519–527, 2012.

[113] M. Jiang. The zero exemplar distance problem. *Journal of Computational Biology*, 18(9):1077–1086, 2011.

[114] W. Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology*, 8(6):615–623, 2001.

[115] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

[116] C. Komusiewicz and R. Niedermeier. New races in parameterized algorithmics. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS '12)*, volume 7464 of *LNCS*, pages 19–30. Springer, 2012.

[117] S. Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113:58–97, 2014.

[118] T. Lee, J. C. Na, H. Park, K. Park, and J. S. Sim. Finding consensus and optimal alignment of circular strings. *Theoretical Computer Science*, 468:92–101, 2013.

[119] H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

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

[121] D. Lokshtanov, D. Marx, and S. Saurabh. Slightly superexponential parameterized problems. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '11)*, pages 760–776. SIAM, 2011.

[122] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–71, 2011.

[123] D. P. Lopresti and A. Tomkins. Block edit models for approximate string matching. *Theoretical Computer Science*, 181(1):159–179, 1997.

[124] B. Ma and X. Sun. More efficient algorithms for closest string and substring problems. *SIAM Journal on Computing*, 39(4):1432–1443, 2009.

[125] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999.

[126] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(3):322–336, 1978.

[127] D. Marx. Closest substring problems with small distances. *SIAM Journal on Computing*, 38(4):1382–1410, 2008.

[128] D. Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

[129] D. Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Operations Research Letters*, 36(1):31–36, 2008.

[130] D. Marx. Fixed-parameter tractable scheduling problems. In *Packing and Scheduling Algorithms for Information and Communication Services (Dagstuhl Seminar 11091)*. 2011.

[131] D. Meister. Using swaps and deletes to make strings match. Technical Report 14-1, Fachbereich IV, Universität Trier, May 2014.

[132] M. Middendorf. More on the complexity of common superstring and supersequence problems. *Theoretical Computer Science*, 125(2):205–228, 1994.

[133] M. Middendorf. Shortest common superstrings and scheduling with coordinated starting times. *Theoretical Computer Science*, 191(1–2): 205–214, 1998.

[134] M. Mnich and A. Wiese. Scheduling and fixed-parameter tractability. In *17th International Conference on Integer Programming and Combinatorial Optimization (IPCO '14)*, volume 8494 of *LNCS*, pages 381–392. Springer, 2014.

[135] C. Moan and I. Rusu. Hard problems in similarity searching. *Discrete Applied Mathematics*, 144(1-2):213–227, 2004.

[136] N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh. LP can be a cure for parameterized problems. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS '12)*, LIPIcs, pages 338–349. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

[137] F. Nicolas and E. Rivals. Hardness results for the center and median string problems under the weighted and unweighted edit distances. *Journal of Discrete Algorithms*, 3(2-4):390–415, 2005.

[138] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, February 2006.

[139] R. Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS '10)*, volume 5 of *LIPIcs*, pages 17–32. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010.

[140] N. Nishimura and N. Simjour. Enumerating neighbour and closest strings. In *Proceedings of the 7th International Symposium on Parameterized and Exact Computation (IPEC '12)*, volume 7535 of *LNCS*, pages 252–263. Springer, 2012.

[141] P. A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. The MIT Press, 2000.

[142] K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003.

[143] W. Plandowski. Satisfiability of word equations with constants is in PSPACE. *Journal of the ACM*, 51(3):483–496, 2004.

[144] A. Radcliffe, A. Scott, and E. Wilmer. Reversals and transpositions over finite alphabets. *SIAM Journal on Discrete Mathematics*, 19(1): 224–244, 2005.

[145] D. Reidenbach and M. L. Schmid. Patterns with bounded treewidth. *Information and Computation*, 2014. Available online.

[146] S. S. Skiena. *The Algorithm Design Manual*. Springer, 2nd edition, 2010.

[147] K. M. Swenson, M. Marron, J. V. Earnest-DeYoung, and B. M. E. Moret. Approximating the true evolutionary distance between two genomes. *ACM Journal on Experimental Algorithmics*, 12, 2008.

[148] V. G. Timkovskii. Complexity of common subsequence and supersequence problems and related problems. *Cybernetics*, 25(5):565–580, 1989.

[149] Y.-T. Tsai. The constrained longest common subsequence problem. *Information Processing Letters*, 88(4):173–176, 2003.

[150] V. Vassilevska. Explicit inapproximability bounds for the shortest superstring problem. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS '05)*, volume 3618 of *LNCS*, pages 793–800. Springer, 2005.

[151] R. A. Wagner. On the complexity of the extended string-to-string correction problem. In *Proceedings of the 7th ACM Symposium on Theory of Computing (STOC '75)*, pages 218–223. ACM, 1975.

[152] L. Wang and B. Zhu. Efficient algorithms for the closest string and distinguishing string selection problems. In *Proceedings of the 3rd International Workshop on Frontiers in Algorithmics (FAW '09)*, volume 5598 of *LNCS*, pages 261–270. Springer, 2009.

[153] C. Zheng, Q. Zhu, and D. Sankoff. Removing noise and ambiguities from comparative maps in rearrangement analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):515–522, 2007.

[154] P. Zörnig. Improved optimization modelling for the closest string and related problems. *Applied Mathematical Modelling*, 35(12):5609–5617, 2011.