String Factorizations Under Various Collision Constraints

Niels Grüttemeier
Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany
niegru@informatik.uni-marburg.de

Christian Komusiewicz
Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany
komusiewicz@informatik.uni-marburg.de

Nils Morawietz
Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany
morawietz@informatik.uni-marburg.de

Frank Sommer
Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany
fsommer@informatik.uni-marburg.de

Abstract

In the NP-hard Equality-Free String Factorization problem, we are given a string S and ask whether S can be partitioned into k factors that are pairwise distinct. We describe a randomized algorithm for Equality-Free String Factorization with running time $2^k \cdot k^{\mathcal{O}(1)} + \mathcal{O}(n)$ improving over previous algorithms with running time $k^{\mathcal{O}(k)} + \mathcal{O}(n)$ [Schmid, TCS 2016; Mincu and Popa, Proc. SOFSEM 2020]. Our algorithm works for the generalization of Equality-Free String Factorization where equality can be replaced by an arbitrary polynomial-time computable equivalence relation on strings. We also consider two factorization problems to which this algorithm does not apply, namely Prefix-Free String Factorization where we ask for a factorization of size k such that no factor is a prefix of another factor and Substring-Free String Factorization where we ask for a factorization of size k such that no factor is a substring of another factor. We show that these two problems are NP-hard as well. Then, we show that Prefix-Free String Factorization with the prefix-free relation is fixed-parameter tractable with respect to k by providing a polynomial problem kernel. Finally, we show a generic ILP formulation for R-Free String Factorization where R is an arbitrary relation on strings. This formulation improves over a previous one for Equality-Free String Factorization in terms of the number of variables.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Pattern matching

Keywords and phrases NP-hard problem, fixed-parameter algorithms, collision-aware string partitioning

Digital Object Identifier 10.4230/LIPIcs.CPM.2020.17

Funding *Frank Sommer*: Supported by the Deutsche Forschungsgemeinschaft (DFG), project MAGZ, KO 3669/4-1.

1 Introduction

In collision-aware string partitioning problems we are given a string S and want to compute a factorization of S, that is, a partition of S into substrings, called factors, such that no two factors of the factorization collide. Herein, two strings collide if they are too similar, for example if they are equal or if one is a prefix of the other [8]. These problems have applications in synthetic biology, where one important task is to assemble a DNA string S from some of its factors. To allow for an assembly of S, the factors from which S is built need

to be sufficiently different from each other, as otherwise the assembly process will produce some unwanted string $S' \neq S$. The demand for pairwise inequality or pairwise prefix-freeness of the factors is an abstraction of the demand of sufficiently large differences [8]. Such demands are always fulfilled by the trivial factorization which consists of the single factor S but in the application described above, we aim to find small factors. Thus, the task is to find a factorization with the desired property such that each factor has bounded length [7, 8]. Another closely related variant of collision-aware string partitioning arises in the context of pattern matching with variables [11]. Here, the additional restriction is not on the length of the factors but instead on their number.

Equality-Free String Factorization
Input: A string S of length n and an integer k.
Question: Is there a factorization of S into k pairwise different factors?

Equality-Free String Factorization is NP-hard [11]. Motivated by this result, Schmid [16] initiated a parameterized complexity analysis with respect to parameters such as the alphabet size of S or the factorization size k. For the latter parameter, an algorithm with running time $\mathcal{O}((\frac{k^2+k}{2}-1)^k + n)$ was proposed [16]. This algorithm relies on a combination of the brute-force algorithm with running time $\mathcal{O}(n^k)$ with the observation that instances with $n \geq \frac{k^2+k}{2}-1$ are yes-instances. The running time was later improved to $\mathcal{O}(k^{k/2}+n)$ [15]. We continue the study of Equality-Free String Factorization with respect to the natural parameter k. Moreover, we consider several extensions and variants of Equality-Free String Factorization and study their classical and parameterized complexity.

Our Results. We present an improved randomized fixed-parameter algorithm for Equality-Free String Factorization with a running time of $2^k \cdot n^{\mathcal{O}(1)}$. This algorithm relies on a reduction to the problem of finding a path with k different colors in a directed graph G and on an algebraic algorithm for finding such a path. This is one of the few applications of algebraic algorithms to NP-hard string problems, another application was provided for Maximum Duo String Partitioning [13]. Curiously, in both applications the first step is a reduction to a path-finding problem in an auxiliary graph. Unlike previous approaches which are tailored to Equality-Free String Factorization since they use the fact that strings with different length are unequal, our algorithm works for an arbitrary equivalence relation R over strings and thus for further notions of collision. To formulate our result precisely, we introduce the following generic problem.

R-Free String Factorization
Input: A string S of length n and an integer k.
Question: Is there a factorization $(w_1, w_2, \ldots, w_k)$ of S such that $w_i \not R\ w_j$ for all $i \neq j$?

▶ **Theorem 1.1.** *Let R be a polynomial-time computable equivalence relation over the set of all strings. Then, R-Free String Factorization can be solved by a randomized algorithm with one-sided error and running time $2^k \cdot n^{\mathcal{O}(1)}$.*

Two natural examples for such an equivalence relation are to consider two strings as similar when they use the same set of letters, we denote this relation by $=_\Sigma$, and to consider two strings w and w' as similar when they have the same Parikh vector, we denote this relation by $=_{\Sigma,\#}$. The Parikh vector of a string w over alphabet $\Sigma = \{a_1, \ldots, a_\sigma\}$ is the length-σ vector p_w where $p_w[i]$ is the number of occurrences of a_i in w. This notion of equivalence is used in Jumbled Pattern Matching [6]. Since $=_\Sigma$ and $=_{\Sigma,\#}$ are equivalence

relations, Theorem 1.1 directly implies an FPT algorithm for them. It is a priori not clear, however, whether R-Free String Factorization is even NP-hard for these particular special cases of R. To show NP-hardness of these two special cases, we revisit the NP-hardness proof for Equality-Free String Factorization [11].

Motivated by the different notions of string collision that have been formulated previously [7, 8] we then consider three cases of R that are not equivalence relations: In the *prefix relation* $\preccurlyeq_p$ we have $w \preccurlyeq_p w'$ if w is a prefix of w', in the *suffix relation* $\preccurlyeq_s$ we have $w \preccurlyeq_s w'$ if w is a suffix of w', and in the *substring relation* $\preccurlyeq$ we have $w \preccurlyeq w'$ if w is a substring of w'. By slightly adapting the known hardness reduction for Equality-Free String Factorization [11], we obtain the following.

▶ **Theorem 1.2.** *For each $R \in \{=_\Sigma, =_{\Sigma,\#}, \preccurlyeq_p, \preccurlyeq_s, \preccurlyeq\}$, R-Free String Factorization is NP-complete and cannot be solved in time $2^{o(n)}$ unless the ETH fails.*

Our second main technical result is a problem kernel for Prefix-Free String Factorization, that is, for the special case where R is the prefix relation $\preccurlyeq_p$. This kernel proves that Prefix-Free String Factorization is fixed-parameter tractable for the parameter k despite the fact that Theorem 1.1 does not apply. The main idea of the kernelization is to shrink highly repetitive regions of the string S. Moreover, this result also implies that Suffix-Free String Factorization, which is the special case of R-Free String Factorization where $R = \preccurlyeq_s$, is fixed-parameter tractable for the parameter k. Finally, as a side result we obtain an ILP formulation with $\mathcal{O}(n)$ variables for R-Free String Factorization for all relations R that can be computed in polynomial time. This improves, in terms of the number of variables, upon a previous formulation for Equality-Free String Factorization [15].

Related Work. Fernau et al. [11] introduced the problem of maximizing the number of factors and showed that it is NP-hard. The NP-hardness reduction also implies that, assuming the ETH, Equality-Free String Factorization cannot be solved in $2^{o(n)}$ time (this uses the fact that 3D Matching cannot be solved in $2^{o(q)}$ time, where q is the instance size [12]). Mincu and Popa [15] introduced a version of Equality-Free String Factorization in which one allows gaps between the factors. That is, the aim is to find k disjoint factors of S such that no two are equal. A further related NP-hard factorization problem is Diverse Palindromic Factorization, where we ask whether a given string has an equality-free factorization in which each factor is a palindrome [1].

Preliminaries. For $i \in \mathbb{N}$, we let $[i]$ denote the set $\{1, \ldots, i\}$. For $i \in \mathbb{N}$ and $j \in \mathbb{N}$, where $i \leq j$, we let $[i, j]$ denote the set $\{i, \ldots, j\}$.

The length of a string S is denoted by $|S|$. For a string S, we let $S[i]$, $1 \leq i \leq |S|$, denote the character at position i and $S[i, j]$, $1 \leq i \leq j \leq |S|$, denote the substring starting at position i and ending at position j. Given a string $S = S[1]S[2] \ldots S[|S|]$, we define $S^1 := S$ and $S^i := S^{i-1}S$ for $i > n$. Moreover, we let $\overleftarrow{S} := S[|S|]S[|S|-1] \ldots S[1]$ denote the *reversed string of* S. A *period* of S is an integer p such that $S[i] = S[i+p]$ for all $i \in [|S| - p]$. The string $S[1, p]$ is also called period in this case. If a string S can be written as $w'w^iw''$ for some $i \geq 1$ and has period $|w|$, then we call w an *internal period* of S. A *border* of a string S is a suffix of S that is also a prefix of S. If a string has a border of length b, then it has a period of length $|S| - b$ [9]. Two substrings $S[i, j]$ and $S[i', j']$ *overlap* if $[i, j] \cap [i', j'] \neq \emptyset$, otherwise they are *disjoint*. A set $\mathcal{P}$ of substrings of a string S is a *packing* if all substrings in $\mathcal{P}$ are disjoint. A set $\mathcal{S}$ of strings is *prefix-free* if no string in $\mathcal{S}$ is a prefix of another string in $\mathcal{S}$. To avoid confusion, we will use the term *factor* only in combination with a factorization and not as a synonym of substring.

We consider directed graphs $D = (V, A)$ where V is a set of vertices and $A \subseteq V \times V$ is a set of directed edges called *arcs*. A *walk* of length k in a directed graph $D = (V, A)$ is a k-tuple $(v_1, \ldots, v_k)$ such that $v_i \in V$ for each $i \in [k]$ and $(v_i, v_{i+1}) \in A$ for each $i \in [k-1]$. A walk $(v_1, \ldots, v_k)$ is a (simple) *path* if $v_i \neq v_j$ for all $i \neq j$. A walk $(v_1, \ldots, v_k)$ is a *cycle* whenever $v_1 = v_k$.

For more details on parameterized algorithms and the Exponential Time Hypothesis (ETH), we refer the reader to the standard monographs. For an overview of the parameterized complexity of string problems, refer to the survey of Bulteau et al. [5]. For an introduction to algebraic algorithms including evaluation of polynomials over finite fields, refer to the monograph of Cygan et al. [10].

Due to lack of space, several proofs are deferred to the long version of this paper.

Lower-Bounding the Factorization Size. In the definition of R-Free String Factorization we ask for a factorization into *exactly* k factors. An equally natural question would be to ask for a factorization into *at least* k factors. It is known that for Equality-Free String Factorization these two questions are equivalent: by merging a largest factor with a neighboring factor, any equality-free factorization of size k' can be transformed into one of size $k' - 1$. Such a property is also possible for the relations R under consideration in this work: For $=_\Sigma$, merge a factor with a maximal set of characters with one of its neighboring factors. For $=_{\Sigma,\#}$, merge the factor with the lexicographically largest Parikh vector with one of its neighboring factors. For $\preccurlyeq_p$, we make use of the following observation which will also be useful in the kernelization algorithm.

▶ **Lemma 1.3.** *Let $\mathcal{S}$ be a prefix-free set of strings over an alphabet Σ and let $u \in \mathcal{S}$ be any string of $\mathcal{S}$ and let $v \in \Sigma^*$. Then the set $\mathcal{S} \setminus \{u\} \cup \{uv\}$ is prefix-free.*

Proof. The string uv is not a prefix of any other string in $\mathcal{S}$ since the string u is not a prefix of any other string in $\mathcal{S}$. Moreover, no other string w is a prefix of uv: Otherwise, if $|w| \leq |u|$, then w is a prefix of u and if $|w| > u$, then u is a prefix of w. In both cases, we have a contradiction to the fact that $\mathcal{S}$ is prefix-free. ◀

Lemma 1.3 can now be used to argue that asking for a prefix-factorization of size k is equivalent to asking for one of size at least k: Given a prefix-free factorization $(f_1, \ldots, f_{k'})$ where $k' > 1$, merge f_1 and f_2 into one factor. By Lemma 1.3 the factorization $(f_1 f_2, \ldots, f_{k'})$ is prefix-free and has size size $k' - 1$.

2 An Improved Parameterized Algorithm

A Reduction to a Rainbow Path Problem. The first step in the improved algorithm is to reduce R-Free String Factorization to the following path problem.

Rainbow-(s,t)-Path
Input: A directed graph $D = (V, A)$, two vertices $s \in V$ and $t \in V$, and a vertex-coloring $c : V \to \{1, \ldots, |V|\}$.
Question: Does G contain a *rainbow (s,t)-path* of length k, that is, a path on k vertices from s to t such that all vertices on this path have pairwise different colors?

▶ **Lemma 2.1.** *There is a parameterized polynomial-time reduction from R-Free String Factorization parameterized by k to Rainbow-(s,t)-Path parameterized by k. Instances with parameter value k are mapped to instances with parameter value $k + 2$ and the graph produced by the reduction is a DAG.*

Proof. For each substring $S[i,j]$, create a vertex $v_{i,j}$. For each vertex $v_{i,j}$ add an arc to each vertex $v_{j+1,q}$, $j+1 \leq q \leq n$. In other words, arcs are added between vertices that represent neighboring substrings. For each equivalence class C of R that has at least one representative in the set of substrings of S, we introduce one color c_C. Each vertex $v_{i,j}$ is colored with the color of its equivalence class $c_{[v_{i,j}]}$. Finally, we add one vertex s with a unique color c_s and a vertex t with a unique color c_t and the arcs $(s, v_{1,j})$, $1 \leq j \leq n$, and $(v_{p,n}, t)$, $1 \leq p \leq n$.

The correctness of the reduction can be seen as follows. If S has a size-k factorization F such that for all factors w and w' of F we have $w \not R\, w'$, then D has a rainbow (s,t)-path of length $k+2$: The vertices corresponding to the factors of F form a path of length k and s has an arc to the first vertex of the path and the last vertex of the path has an arc to t. Moreover, the vertices on the path have pairwise different colors since each color corresponds to an equivalence class of R. Conversely, any (s,t)-path in D corresponds to a factorization of S. Moreover, if the vertices of the factorization have a different color, then the corresponding factors are in different equivalence classes and thus not in relation with respect to R. If the length of the path is $k+2$, then the number of factors in the factorization is k. ◀

Observe that one approach to obtain a parameterized algorithm for Rainbow-(s,t)-Path parameterized by k is via color coding: randomly map the $\mathcal{O}(n^2)$ colors in D to a set of k labels and then find an (s,t)-path with k different labels, if it exists, via dynamic programming. Using standard derandomization techniques [10], this algorithm can be derandomized with a slight running time overhead, resulting in a deterministic algorithm with running time $(2e)^k \cdot k^{\mathcal{O}(\log k)} \cdot n^{\mathcal{O}(1)}$, improving over the previous fastest algorithm of Mincu and Popa [15]. We omit the description of this algorithm in favor of a substantially faster randomized algorithm.

Detecting Rainbow (s,t)-Paths. To solve Rainbow-(s,t)-Path, we reduce it to the problem of testing whether a polynomial can be evaluated to zero in some finite field. This technique has led to the currently fastest randomized algorithms for several longest path problems parameterized by the path length [2, 3]. We adapt the technique to handle the constraint that the path should be rainbow. Observe that there is a previous algorithm for Rainbow-(s,t)-Path on undirected graphs [14] which uses the number of colors as parameter. Since in our application the number of colors can be superlinear in n, we may not use this algorithm to obtain a fixed-parameter algorithm for Equality-Free String Factorization.

The algorithm to detect rainbow (s,t)-paths is a slight adaption of the algorithm for Longest Path in directed graphs [10]: Associate a set of monomials (that is, a polynomial with only one summand) with every walk of length k of the graph and then consider the walk polynomial which is the sum of the walk monomials. More precisely, we will introduce one monomial $M_{W,\ell}$ for each walk $W = (v_1, \ldots, v_k)$ and each bijective labeling $\ell : [k] \to [k]$. The labeling ℓ will assign a label $\ell(i)$ to the ith color occurring in the walk and is the main trick to establish the canceling property for walks that are not rainbow. If we ensure that nonsimple walks cancel out, then the polynomial will not be identically zero if and only if there is at least one rainbow path of length k.

For each edge (u,v) we introduce a variable $x_{u,v}$ which represents that the edge (u,v) is traversed in a walk. For each color c of G, we introduce k variables $y_{c,i}$, $i \in [k]$, each representing that a vertex with color c receives the label i. The monomial associated with the pair (W, ℓ) is now

$$M_{W,\ell}(\mathbf{x}, \mathbf{y}) := \prod_{i=1}^{k-1} x_{v_i, v_{i+1}} \prod_{i=1}^{k} y_{c(v_i),\ell(i)}.$$

To ensure that the path starts with s and ends in t, we consider only walks that start in s and end in t. Accordingly, the polynomial is defined as

$$P(\mathbf{x},\mathbf{y}) := \sum_{\text{walk } W=(s=v_1,\ldots,v_k=t)} \sum_{\text{bijective } \ell:[k]\to[k]} M_{W,\ell}(\mathbf{x},\mathbf{y}).$$

As mentioned above, the idea of the construction is that the monomials for walks which have some label twice will cancel out. To enable this, the polynomial is evaluated over a field of characteristic 2, that is, addition of some value to itself will give 0. Thus, the walks that are not paths will cancel out if their monomials can be partitioned into pairs such that each pair will have the same variables. Observe that the polynomial will never be constructed explicitly but instead evaluated via dynamic programming.

▶ **Lemma 2.2.** *$P(\mathbf{x},\mathbf{y})$ is not identically zero in a finite field with characteristic 2 if and only if there is a rainbow (s,t)-path of length k in G.*

Proof. Consider an (s,t)-walk $W = (s = v_1,\ldots,v_k = t)$ that is not rainbow and any monomial $M_{W,\ell}$. Since W is not rainbow, there are two vertices v_i and v_j such that $c(v_i) = c(v_j)$. Take the lexicographically smallest pair of indices i and j for which this is true. Consider the labeling $\ell_{i\leftrightarrow j}$ defined as follows: $\ell_{i\leftrightarrow j}(i) := \ell(j)$, $\ell_{i\leftrightarrow j}(j) := \ell(i)$, and $\ell_{i\leftrightarrow j}(q) := \ell(q)$ for all $q \in [k] \setminus \{i,j\}$. In other words, $\ell_{i\leftrightarrow j}$ is obtained by swapping the ith and jth elements in the permutation corresponding to ℓ. The monomial $M_{W,\ell_{i\leftrightarrow j}}$ is the same as $M_{W,\ell}$ and, hence, $M_{W,\ell} + M_{W,\ell_{i\leftrightarrow j}}$ is identically zero. Moreover, since $(\ell_{i\leftrightarrow j})_{i\leftrightarrow j} = \ell$, we have a partition of all monomials corresponding to walks that are not rainbow paths into pairs $\{\ell, \ell_{i\leftrightarrow j}\}$ such that for each pair, the variables of the monomial are the same. Thus, these monomials cancel out and P can be written as the sum over all walks that are in fact rainbow paths.

It remains to show that the monomials that correspond to rainbow (s,t)-paths do not cancel out, by showing that they have pairwise different variable sets. This is obvious for two monomials that correspond to different walks. Thus, consider a rainbow (s,t)-path W and two monomials $M_{W,\ell}$ and $M_{W,\ell'}$ where ℓ and ℓ' are two different labelings. Moreover, choose $i \in [k]$ such that $\ell(i) \neq \ell'(i)$. Then, $M_{W,\ell}$ and $M_{W,\ell'}$ differ in at least two variables: the variable $y_{c(v_i),\ell(i)}$ occurs in $M_{W,\ell'}$ and not in $M_{W,\ell'}$: First, $y_{c(v_i),\ell'(i)}$ is a different variable since $\ell(i) \neq \ell'(i)$ and each other y-variable in $M_{W,\ell'}$ is of the form $y_{c',q}$ for some $c' \neq c(v_i)$ since W is rainbow. ◀

The polynomial $P(\mathbf{x},\mathbf{y})$ can be efficiently evaluated via dynamic programming.

▶ **Lemma 2.3.** *The polynomial $P(\mathbf{x},\mathbf{y})$ can be evaluated in $2^k \cdot k^{\mathcal{O}(1)} \cdot |A|$ time over the field $\mathrm{GF}(2^{\lceil \log 4k \rceil})$.*

Proof. We follow the exposition of Cygan et al. [10] and present the details only for the sake of completeness. Fix a walk W, then the sum of the monomials $M_{W,\ell}$ over all bijective labelings ℓ can be written as

$$\sum_{\text{surjective } \ell:[k]\to[k]} M_{W,\ell} = \sum_{\ell \in \bigcap_{i\in[k]} A_i} M_{W,\ell}(\mathbf{x},\mathbf{y})$$

where A_i is the set of labelings such that $\ell(j) = i$ for some $j \in [k]$. Now, let U denote the set of all mappings $\ell : [k] \to [k]$. Using the inclusion–exclusion principle, the latter term can

be rewritten as.

$$\begin{aligned}\sum_{\ell \in \bigcap_{i\in[k]} A_i} M_{W,\ell}(\mathbf{x},\mathbf{y}) &= \sum_{X\subseteq[k]} (-1)^{|X|} \cdot \sum_{\ell \in \bigcap_{i\in X} U\setminus A_i} M_{W,\ell}(\mathbf{x},\mathbf{y}) \\ &= \sum_{X\subseteq[k]} \sum_{\ell \in \bigcap_{i\in X} U\setminus A_i} M_{W,\ell}(\mathbf{x},\mathbf{y}) \\ &= \sum_{X\subseteq[k]} \sum_{\ell:[k]\to[k]\setminus X} M_{W,\ell}(\mathbf{x},\mathbf{y}) \\ &= \sum_{X\subseteq[k]} \sum_{\ell:[k]\to X} M_{W,\ell}(\mathbf{x},\mathbf{y}).\end{aligned}$$

The equalities follow from the inclusion–exclusion principle, the fact that the field has characteristic 2, the fact that $\bigcap_{i\in X} U \setminus A_i$ is the set of all labelings $\ell : [k] \to [k]$ that do not map to any element in X, and a replacement of X by its complement $U \setminus X$ in the sum, respectively.

Thus, we have

$$\begin{aligned}P(\mathbf{x},\mathbf{y}) &= \sum_{\text{walk } W=(s=v_1,\ldots,v_k=t)} \sum_{\text{bijective } \ell:[k]\to[k]} M_{W,\ell}(\mathbf{x},\mathbf{y}) \\ &= \sum_{\text{walk } W=(s=v_1,\ldots,v_k=t)} \sum_{X\subseteq[k]} \sum_{\ell:[k]\to X} M_{W,\ell}(\mathbf{x},\mathbf{y}) \\ &= \sum_{X\subseteq[k]} \sum_{\text{walk } W=(s=v_1,\ldots,v_k=t)} \sum_{\ell:[k]\to X} M_{W,\ell}(\mathbf{x},\mathbf{y})\end{aligned}$$

It is thus sufficient to show that $\sum_{\ell:[k]\to X} M_{W,\ell}(\mathbf{x},\mathbf{y})$ can be computed in $k^{\mathcal{O}(1)}\cdot|A|$ time. This can be done by dynamic programming, building up the domain of the labeling ℓ from [1] to $[k]$. More precisely, one may fill a table with entries of the type $T_X[v,d]$ where v is a vertex of D and $d \in [k]$ such that

$$T_X[v,d] := \sum_{\text{walk } W=(v=v_1,\ldots,v_d=t)} \sum_{\ell:[d]\to X} M_{W,\ell}(\mathbf{x},\mathbf{y})$$

as follows. For each the vertex t, $T[t,1] = y_{c(t),i}$ as the walk (t) does not contain any edges, and we may consider all possible labels for $y_{c(t),\ell}$. For $d > 1$, we have

$$T_X[v,d] = \sum_{i\in X} y_{c(v),i} \sum_{(v,w)\in A} x_{v,w} \cdot T[w,d-1]$$

since this sum considers all possibilities for the label of $c(v)$, all outgoing edges from v, and multiplies each with the number of possibilities to continue the walk. Here, we exploit that $T[w,d]$ is not just the sum over all walks and all labelings $\ell : [d] \to X$ but due to symmetry, the sum over all walks and all labelings $\ell : X' \to X$ for every $X' \subseteq X$ such that $|X'| = d$.

The value of $P(\mathbf{x}, \mathbf{y})$ can thus be computed using $\mathcal{O}(2^k \cdot k|A|)$ field operations since

$$P(\mathbf{x}, \mathbf{y}) = \sum_{X \subseteq [k]} \sum_{\text{walk } W=(s=v_1,\ldots,v_k=t)} \sum_{\ell:[k]\to X} M_{W,\ell}(\mathbf{x}, \mathbf{y}) \tag{1}$$

$$= \sum_{X \subseteq [k]} T_X[s, k]. \tag{2}$$

Since the order of the field is $2^{\lceil \log 4k \rceil} \leq 8k$, each field operation can be performed in $k^{\mathcal{O}(1)}$ time. ◀

Now we obtain a randomized algorithm for deciding whether D contains a rainbow (s, t)-path by evaluating P at a random vector $(\mathbf{x}, \mathbf{y})$ of elements of the field $\mathrm{GF}(2^{\lceil \log 4k \rceil})$. If $P(\mathbf{x}, \mathbf{y}) = 0$, then the algorithm returns that D has no rainbow (s, t)-path of length k, otherwise it returns that D contains such a path. If the graph contains no rainbow (s, t)-path of length k, then P is identically zero and the algorithm answers correctly. Otherwise, if the graph contains a rainbow (s, t)-path of length k, then P is not identically zero. Since the maximum degree of P is at most $2k - 1$ and since the field has order at least $4k$, the Schwartz-Zippel Lemma now implies that $(\mathbf{x}, \mathbf{y})$ is a root with probability at most $1/2$. Thus, the error probability is at most $1/2$. By repeating this procedure $\log(1/\epsilon)$ times, we may achieve an error probability of at most ϵ for any $\epsilon > 0$.

This algorithm in combination with the reduction from R-Free String Factorization to Rainbow-(s, t)-Path gives Theorem 1.1. Observe that Theorem 1.1 only refers to the decision version of the problem but in the applications we may want to output the factorization if it exists. This can be done via applying the framework of Björklund et al. [4] which only relies on two facts: The witness which we wish to extract has size k and the decision algorithm has one-sided error; the running time overhead is a factor of $\mathcal{O}(k \log n)$.

As a final remark, in our framework of solving R-Free String Factorization via reduction to Rainbow-(s, t)-Path, we may put any further polynomial-time computable restriction on the factors: the only additional step is to add only those vertices that fulfill this restriction to the graph D. For example, we may demand that every factor has length at least q and at most r for some integers q and r, or we may demand that it contains every letter of the alphabet. Thus, we may apply the algorithm also when we search for length-bounded factorizations [8] when the parameter is the factorization size k. We can also use this framework to solve Diverse Palindromic Factorization where each factor should be a palindrome [1].

▶ **Corollary 2.4.** *There is a randomized algorithm with one-sided error that decides in time $2^k \cdot n^{\mathcal{O}(1)}$ whether a string has a palindromic factorization with exactly k factors.*

3 Further String Relations

Hardness Results. First, we prove Theorem 1.2 . That is, we show that for each $R \in \{=_\Sigma, =_{\Sigma,\#}, \preccurlyeq_p, \preccurlyeq_s, \preccurlyeq\}$, R-Free String Factorization is NP-complete and cannot be solved in time $2^{o(n)}$ unless the ETH fails.

First, we show this result for $R \in \{\preccurlyeq_p, \preccurlyeq\}$.

▶ **Lemma 3.1.** *For each $R \in \{\preccurlyeq_p, \preccurlyeq\}$, R-Free String Factorization is NP-complete and cannot be solved in time $2^{o(n)}$ unless the ETH fails.*

Observe that the hardness result for $R = \preccurlyeq_p$ also implies hardness for $R = \preccurlyeq_s$ by the following simple reduction: Let (S, k) be an instance of R-Free String Factorization for $R = \preccurlyeq_p$. Then, the instance $(\overleftarrow{S}, k)$ is an equivalent instance for $R = \preccurlyeq_s$. The equivalence can be seen as follows: $(f_1, f_2, \ldots, f_k)$ is a prefix-free string factorization of S if and only if $(\overleftarrow{f_k}, \ldots, \overleftarrow{f_2}, \overleftarrow{f_1})$ is a suffix-free string factorization of $\overleftarrow{S}$. Hence, the following holds.

▶ **Lemma 3.2.** *If $R = \preccurlyeq_s$, then R-Free String Factorization is NP-complete and cannot be solved in time $2^{o(n)}$ unless the ETH fails.*

Next, we show NP-hardness for each $R \in \{=_\Sigma, =_{\Sigma,\#}\}$; altogether this shows Theorem 1.2.

▶ **Lemma 3.3.** *For each $R \in \{=_\Sigma, =_{\Sigma,\#}\}$, R-Free String Factorization is NP-complete and cannot be solved in time $2^{o(n)}$ unless the ETH fails.*

Prefix-Free String Factorization. In this paragraph we provide a problem kernel for Prefix-Free String Factorization parameterized by the number k of factors. Recall that Prefix-Free String Factorization is the special case of R-Free String Factorization, where $R = \preccurlyeq_p$. For Equality-Free String Factorization, it is very easy to obtain a problem kernel: since factors with different length are trivially unequal, one may simply choose strings of increasing length starting with length 1. This implies that instances with $n \geq \frac{k^2+k}{2}$ are yes-instances and thus we have a quadratic kernel for Equality-Free String Factorization. For Prefix-Free String Factorization, this argument does not hold. Consider for example the string a^n. The maximum prefix-free factorization has size one because of the periodicity of a^n.

To describe the kernelization we first need to establish some notation. Let (S, k) be an instance of Prefix-Free String Factorization. For any string w, a substring $S[i, j]$ of S is called *w-periodic* if $S[i, j] = w^t$ for some integer $t \geq 0$ and if $|w|$ is the shortest period of $S[i, j]$. Moreover, $S[i, j]$ is called *maximal w-periodic* if there is no substring $S[i', j'] = w^{t'}$ with $[i, j] \subsetneq [i', j']$. We define $R(w) := \{t \mid w^t \text{ is a maximal } w\text{-periodic substring of } S\}$. The central rule of this kernelization reduces the length of maximal w-periodic substrings of S. For fixed w, the maximal w-periodic substrings of S are uniquely defined since each begins with w and $|w|$ is the shortest period. Hence, for each w we can find all maximal w-periodic substrings in linear time. We first show that we may assume that for every w the size of $R(w)$ is bounded, which we need to give a bound for the kernel size.

▶ **Lemma 3.4.** *Let (S, k) be an instance of Prefix-Free String Factorization. If there exists a string w such that $|R(w)| \geq 2k + 3$, then (S, k) is a yes-instance.*

Proof. Let w be a substring of S such that $|R(w)| \geq 2k + 3$. Then, there are distinct $t_1, t_2, \ldots, t_{2k+2} \in R(w)$ such that $t_i \geq 2$ for each $i \in [2k + 1]$. We show that we can use maximal w-periodic occurrences of the w^{t_i} to find at least k prefix free-factors. To this end, let $X = \{(p_1, q_1), \ldots, (p_{2k+2}, q_{2k+2})\}$ be a set containing the start positions p_i and the end positions q_i of one maximal w-periodic occurrence of w^{t_i} for each $i \in \{1, \ldots, 2k + 2\}$. By the definition of maximal w-periodic substrings no element $S[p_i, q_i]$ includes another element $S[p_j, q_j]$ in S. Hence, $p_i \neq p_j$ if $i \neq j$. Without loss of generality we assume that $p_1 < p_2 < \cdots < p_{2k+2}$.

Note that in S two strings $S[p_i, q_i]$ and $S[p_j, q_j]$ with $i \neq j$ overlap with less than $|w|$ characters, since otherwise this contradicts the fact that these strings are maximal w-periodic. To obtain the prefix-free factors from X, we first show the following.

▷ Claim 3.5. In S, every string $S[p_i, q_i]$ overlaps with at most two other strings $S[p_j, q_j]$, and $S[p_t, q_t]$.

Proof. Let $S[p_i, q_i]$ overlap with $S[p_j, q_j]$, and $S[p_t, q_t]$. To prove the claim we show that this implies that $S[p_j, q_j]$ and $S[p_t, q_t]$ do not overlap in S.

Without loss of generality assume $p_j < p_i < p_t$. Since $S[p_i, q_i]$ overlaps with $S[p_j, q_j]$ and $S[p_t, q_t]$, and do not overlap in at least $|w|$ characters as discussed above, we have $0 \leq q_j - p_i < |w| - 1$ and $0 \leq q_i - p_t < |w| - 1$. Since $t_2 \geq 2$ by the definition of X it also holds that $q_i - p_i \geq 2|w| - 1$ and therefore $q_i - p_i > q_j - p_i + q_i - p_t$. Consequently it holds that $q_j < p_t$ and therefore $S[p_j, q_j]$ and $S[p_t, q_t]$ do not overlap in S. ◁

We next use Claim 3.5 to define the factors. We define $k + 1$ disjoint substrings of S at follows: $f_0 := S[1, p_2 - 1]$, $f_i := S[p_{2i+1}, p_{2i+3} - 1]$ for all $i \in [k - 1]$, and $f_k := S[p_{2k+1}, |S|]$. Claim 3.5 guarantees that $w^{t_{2i+1}}$ is a prefix of f_i if $i \geq 1$. Then, for every distinct $f, f' \in \{f_1, \ldots, f_k\}$, the substring f is not a prefix of f' since f and f' start with substrings of period $|w|$ that have distinct lengths. Next, consider the following cases for f_0.

Case 1: There exists no f_i with $i \geq 1$ such that f_0 is a prefix of f_i or vice versa**.** Then, $(f_0, \ldots, f_k)$ is a prefix-free string factorization of size $k + 1$ and nothing more needs to be shown.

Case 2: There exists some f_i with $i \geq 1$ such that f_0 is a prefix of f_i or vice versa**.** Then, f_0 starts with a maximal w-periodic substring $w^{t_{2i+1}}$. Hence, for every $j \neq i$ it holds that f_0 is not a prefix of f_j and vice versa. We then discard f_{i-1} and f_i and instead consider their concatenation $f_{i-1}f_i$. We end up with a prefix-free factorization of S containing at least k factors. Hence, (S, k) is a yes-instance. ◀

For the rest of this section, we assume that, given an instance (S, k), we have $|R(w)| \leq 2k + 3$ for each substring w of S since otherwise (S, k) is a trivial yes-instance due to Lemma 3.4. The main idea of the kernelization is thus to reduce the number of maximum repetitions of every substring of the input string S.

▶ **Rule 3.1.** *Let w be a substring in S such that S has*
1. *at least one maximal w-periodic substring $S[q, r] = w^d$ with $d \geq 2k^2 + 2$ and*
2. *for each $\psi \in [k^2]$ no maximal w-periodic substring $S[q', r'] = w^{d-\psi}$.*

Then, for each $p \geq d$, replace each maximal w-periodic substring $S[q, r] = w^p$ by w^{p-1}.

The rule can be applied in polynomial time by checking for each substring w of S whether it fulfills the conditions of the rule. The correctness proof of the rule is quite technical and due to lack of space deferred to the long version. The proof idea is as follows. When the rule shortens a maximal w-periodic substring w^p, one of the k factors that overlaps with this factor must be shortened by one occurrence of w. This may, however, lead to a factorization that is not prefix-free. To reestablish prefix-freeness, we may need to remove w from another factor f that overlaps some other maximal w-periodic substring $w^{p'}$. Since $w^{p'}$ is not necessarily shortened by the rule, we must add w to some other factor f' that overlaps $w^{p'}$ in order to reestablish that we have a factorization. Due to Lemma 1.3, we may add w safely to the factor f' that starts before $w^{p'}$ and ends either right before the first position of $w^{p'}$ or overlaps with $w^{p'}$.

Let w be a substring of S. According to Lemma 3.4, there are at most $2k + 2$ different repetition numbers in $R(w)$. Let $d \geq 2k^2+1$ be a repetition number of w such that for each $\psi \in [k^2]$ we have $d-\psi \notin R(w)$. Then Reduction Rule 3.1 reduces each repetition number $p \geq d$ by 1. Hence, the largest repetition number of w is at most $(2k^2+1)+k^2 \cdot (2k+2) = 2k^3+4k^2+1$ if Reduction Rule 3.1 has been applied exhaustively.

▶ **Corollary 3.6.** *Let S be an instance to which Reduction Rule 3.1 has been applied exhaustively and let w be a substring of S. Then the maximal repetition number of w in S is $2k^3 + 4k^2 + 1$.*

We now show that bounding the number of repetition numbers and their size for all substrings of S results in an instance that is a yes-instance if $|S|$ is too big. This implies the problem kernel.

▶ **Theorem 3.7.** *Prefix-Free String Factorization has a problem kernel of size $\mathcal{O}(k^{10})$.*

Proof. Let (S, k) be an instance of Prefix-Free String Factorization that is reduced exhaustively with respect to Rule 3.1. We show that if $|S| > 12k(k+1)(2k^4 + 4k^3 + 2k)^2$, then (S, k) is a yes-instance.

Let (S, k) be a no-instance. Let $d(w)$ be the maximum size of disjoint occurrences of a substring w in S. To show $|S| \leq 12k(k+1)(2k^4 + 4k^3 + 2k)^2$ we prove that for every substring with length $2k^4 + 4k^3 + 2k$ we have $d(w) \leq 2k$. Afterwards, we use this upper bound on the number of occurrences to give an upper bound for the size of S.

Let w be a substring of S such that $|w| = 2k^4 + 4k^3 + 2k$. Moreover, let $\mathcal{P} := \{S[p_1, q_1], S[p_2, q_2], \ldots, S[p_{|\mathcal{P}|}, q_{|\mathcal{P}|}]\}$ with $p_1 < q_1 < p_2 < \cdots < p_{|\mathcal{P}|} < q_{|\mathcal{P}|}$ be a maximum packing of occurrences of w in S. Without loss of generality, we also assume that $S[p_1, q_1]$ is the first occurrence of w in S, since otherwise, we can replace p_1 and q_1 by the start and endpoint of the first occurrence. Assume towards a contradiction that $|\mathcal{P}| \geq 2k + 1$. We define the subpacking

$$\mathcal{P}' := \{S[p_{2i+1}, q_{2i+1}] \mid i \in [0, k]\} \subseteq \mathcal{P}$$

containing $k+1$ elements from $\mathcal{P}$. For every $i \in \{1, \ldots, k-1\}$ we define the substring $V_{2i+1} := S[p_{2i+1} - k, p_{2i+1} - 1]$ which contains the last k characters before p_{2i+1}. Since $|w| = 2k^4 + 4k^3 + 2k$ and $\mathcal{P}'$ contains every second element of $\mathcal{P}$ it holds that no V_{2i+1} overlaps with $S[p_{2i-1}, q_{2i-1}] \in \mathcal{P}'$.

We first show that no occurrence of w starts in some V_{2i+1}. Assume towards a contradiction that there is one such occurrence starting in some V_{2i+1}. Then, since $|w| = 2k^4 + 4k^3 + 2k$ and $|V_{2i+1}| = k$, the string w has a border of size at least $2k^4 + 4k^3 + k$. Hence, w has a period of length at most k [9] and therefore there exists some z such that there is a maximum z-periodic substring z^p with $p \geq 2k^3 + 4k^2 + 2$ of S. Together with Corollary 3.6, this contradicts the fact that (S, k) is reduced exhaustively regarding Rule 3.1. Hence, we can assume that no occurrence of w starts in some V_{2i+1}. In the following case distinction we consider the possible values of p_1 and show that in each case we can define a prefix-free string factorization of size at least k for S, which then contradicts the fact that (S, k) is a no-instance.

Case 1: $p_1 \geq k + 1$. We define the factors $f_{\text{start}} := S[1, p_3 - 2]$, $f_i := S[p_{2i+1} - i, p_{2i+3} - (i+2)]$ for all $i \in [k-1]$, and $f_{\text{end}} := S[p_{2k+1} - k, |S|]$. Note that these $k + 1$ factors cover all of S and w is a substring of each such factor. We next show that none of these factors is the prefix of another factor. To this end, we consider the first occurrence of w in all factors.

Since $p_1 \geq k+1$ and we assumed that $S[p_1, q_1]$ is the first occurrence of w in S we conclude that in f_{start} there is no occurrence of w starting in the first k positions of f_{start}. Next, the fact that no occurrence of w starts in some V_{2i+1} implies that the first occurrence of w starts at position $i + 1$ of each f_i, $i \in [k-1]$. Moreover, in f_{end}, the first occurrence of w starts at position $k + 1$ by the same argument. Since the first occurrence of w starts at distinct positions in each of the factors, no factor is a prefix of one of the other factors.

Hence, S has a prefix-free factorization with $k+1$ factors contradicting the fact that (S,k) is a no-instance.

Case 2: $p_1 \in \{1,2\}$. We define the factors $f_{\text{start}} := S[1, p_5 - 3]$, $f_i := S[p_{2i+1} - i, p_{2i+3} - (i+2)]$ for all $i \in [2, k-1]$, and $f_{\text{end}} := S[p_{2k+1} - k, |S|]$. Intuitively, these are the $k+1$ factors we defined in Case 1, but we concatenated the first two factors. We obtain k factors that cover all of S and w is a substring of each of the factors. Again we prove prefix-freeness by showing that in each pair of factor the first occurrence of w starts at different positions.

In f_{start}, the first occurrence of w starts at position 1 or 2 since $p_1 \in \{1,2\}$. Since no occurrence of w starts in some V_{2i+1}, the first occurrence of w starts at position $i+1$ in each f_i and at position $k+1$ in f_{end}. Observe that since $i \in [2, \ldots, k-2]$ it holds that $i+1 \geq 3$. Again, this contradicts the fact that (S,k) is a no-instance.

Case 3: $p_1 \in [3, k-1]$. We define the factors $f_{\text{start}} := S[1, p_3 - 2]$, $f_i := S[p_{2i+1} - i, p_{2i+3} - (i+2)]$ for all $i \in [k-1] \setminus \{p_1 - 2, p_1 - 1\}$, $f_{\text{merge}} := S[p_{2p_1-3} - (p_1 - 2), p_{2p_1+1} - (p_1 + 1)]$, and $f_{\text{end}} := S[p_{2k+1} - k, |S|]$. Again, these are k factors covering S containing w as substring. It remains to check for each factor, when the first occurrence of w starts.

In f_{start}, the first occurrence of w starts at p_1. In each f_i the first occurrence of i starts at position $i+1$. Note that $i+1 \notin \{p_1, p_1 - 1\}$. In f_{merge}, the first occurrence of w starts at position $p_1 - 1$. Finally, in f_{end}, the first occurrence of w starts at position $k+1$. Again, this contradicts the fact that (S,k) is a no-instance.

Case 4: $p_1 = k$. We define $f_{\text{start}} := S[1, p_3 - 2]$, $f_i := S[p_{2i+1} - i, p_{2i+3} - (i+2)]$ for all $i \in [k-3]$, $f_i := S[p_{2(k-2)+1} - (k-2), p_{2(k-2)+3} - (k-1)]$, and $f_{\text{end}} := S[p_{2k-1} - k, |S|]$. Again, these are k factors covering S containing w as substring. It remains to check for each factor, when the first occurrence of w starts.

The first occurrence of w in f_{start} starts in position k. The first occurrence of w in each f_i starts in position $i+1$ and the first occurrence of w in f_{end} starts at position $k+1$. Again, this contradicts the fact that (S,k) is a no-instance.

Since all cases are contradictory we know that every substring of size $2k^4 + 4k^3 + 2k$ in S has at most $2k$ disjoint occurrences in S. We next use this fact to prove $|S| \leq 12k(k+1)(2k^4 + 4k^3 + 2k)^2$. To this end, let $X := \{w \mid w \text{ is a substring of } S \text{ and } |w| = 2k^4 + 4k^3 + 2k\}$. We first show that $|X| \leq 2(k+1)(2k^4 + 4k^3 + 2k)$. Assume towards a contradiction that $|X| > 2(k+1)(2k^4 + 4k^3 + 2k)$. Let $w \in X$. Observe that, since $|w| = 2k^4 + 4k^3 + 2k$, every occurrence of w in S overlaps with at most $2(2k^4 + 4k^3 + 2k - 1)$ occurrences of other strings in X. Then, since $|X| > 2(k+1)(2k^4 + 4k^3 + 2k)$, there exists a packing of $k+1$ elements of X. Let $\mathcal{X} := \{w_1, \ldots, w_{k+1}\}$ be such packing and for each $i \in [k+1]$, let p_i be the position in S where w_i starts. We then define $k+1$ disjoint substrings of S as follows: $f_1 := S[1, p_2 - 1]$, $f_i := S[p_i, p_{i+1} - 1]$ for $i = 2, \ldots, k$, and $f_{k+1} := S[p_k, |S|]$. Clearly, for every distinct $f, f' \in \{f_2, f_3, \ldots f_k, f_{k+1}\}$, the substring f is not a prefix of f' since f and f' start with distinct words of length $2k^4 + 4k^3 + 2k$. Consider f_1 and f_i for $i > 1$. If f_1 is a prefix of f_i or vice versa, we discard f_{i-1} and f_i and instead consider their concatenation $f_{i-1}f_i$. Since $|f_1| \geq 2k^4 + 4k^3 + 2k$ we end up with a prefix-free factorization of S containing at least k factors which contradicts the fact that (S,k) is a no-instance. Hence, $|X| \leq 2(k+1)(2k^4 + 4k^3 + 2k)$.

We can now give a bound for $|S|$. Recall that $d(w)$ is the maximum size of disjoint occurrences of w in S and that $d(w) \in [2k]$ for every $w \in X$. Given a fixed $w \in X$, for each of the $d(w)$ disjoint occurrences $S[j, j + |w| - 1]$ of w in S there can be occurrences of w in $S[j - |w|, j + 2|w|]$ that overlap with $S[j, j + |w| - 1]$. Hence, for every $w \in X$, the number of symbols of S in occurrences of w is at most $3|w| \cdot d(w)$. It then holds that $|S| \leq \sum_{w \in X} 3|w| \cdot d(w) \leq 12k(k+1)(2k^4 + 4k^3 + 2k)^2 \in \mathcal{O}(k^{10})$. ◀

The above problem kernelization for Prefix-Free String Factorization also implies a problem kernelization for Suffix-Free String Factorization. To compute a problem kernel for an instance (S, k), we first apply the problem kernelization for Prefix-Free String Factorization on $(\overleftarrow{S}, k)$. Reversing the string of the resulting instance once more gives an instance of the original problem of size $\mathcal{O}(k^{10})$.

▶ **Corollary 3.8.** *Suffix-Free String Factorization has a problem kernel of size* $\mathcal{O}(k^{10})$.

4 An ILP formulation with $\mathcal{O}(n)$ variables

As a side result, we provide an ILP formulation which is better than a previous one in terms of the number of variables. Mincu and Popa [15] described a 0/1-ILP with $\mathcal{O}(n\sqrt{n})$ variables based on the following idea: Introduce a binary variable $x_{i,j}$ for each candidate factor $S[i, j]$ of S such that $x_{i,j} = 1$ precisely if the factor $S[i, j]$ is one of the factors of the factorization. The goal is to maximize $\sum_{1 \leq i \leq j \leq n} x_{i,j}$. The constraints of the ILP ensure that no two equal factors are chosen, no two overlapping factors are chosen, and that if we choose some factor $x_{i,j}$ where $j < n$, then a factor $x_{j+1,\ell}$ must be chosen as well. The number of variables is $\mathcal{O}(n\sqrt{n})$ since, due to an observation of Mincu and Popa [15], there is an optimal equality-free string factorization with factors of length $\mathcal{O}(\sqrt{n})$. The latter observation does not hold for other equivalence relations. For example, if we consider $=_\Sigma$ on binary strings, then we may have at most three factors and thus some factor must have length $\Omega(n)$.

We provide an alternative 0/1-ILP that has $\mathcal{O}(n)$ variables and works for all string relations. For each $i \in \{1, \ldots, n+1\}$, we introduce one variable x_i. This variable will have the value 1 if some factor starts at $S[i]$, that is, the factorization cuts between positions $i-1$ and i. The variable x_1 will correspond to the start of the first factor and the variable x_{n+1} will correspond to the end of the last factor, these variables will be set to 1 and we just introduce them to make the formulation more concise. The whole ILP reads as follows.

$$\text{max.} \sum_{i \in [n+1]} x_i \text{ subject to} \tag{3}$$

$$-x_i - x_{j+1} - x_p - x_{q+1} + \sum_{\ell \in [i+1,j] \cup [p+1,q]} x_\ell > -4 \ \forall i < j < p < q \text{ where } \big(S[i,j] \ R \ S[p,q]\big) \tag{4}$$

$$x_i \in \{0, 1\} \ \forall i \in [2, n] \tag{5}$$

$$x_i = 1 \ \forall i \in \{1, n+1\} \tag{6}$$

Every assignment to the variables directly corresponds to the factorization of the input string where $x_i = 1$ means that some factor starts at position i. The number of factors is one less than the objective function value. It remains to show that no two factors of the factorization are in relation. This is ensured by Constraint 4. Let $S[i, j]$ and $S[p, q]$ be nonoverlapping equivalent candidate factors. Then Constraint 4 for this index set is fulfilled when either one of x_i, x_{j+1}, x_p, and x_{q+1} has value 0, or when one of the variables in the sum has value 1. In the first, case one of the two strings $S[i, j]$ and $S[p, q]$ is not a factor of the factorization, since one of the four factor endpoints is not selected by the solution. In the second case, one of the two candidate factors is not part of the solution since some factor starts after i and before j or after p and before q. Thus, the factorization produced by the ILP is equality-free. Conversely, any equality-free factorization corresponds to a feasible solution of the ILP.

References

1 Hideo Bannai, Travis Gagie, Shunsuke Inenaga, Juha Kärkkäinen, Dominik Kempa, Marcin Piatkowski, and Shiho Sugimoto. Diverse palindromic factorization is NP-complete. *Int. J. Found. Comput. Sci.*, 29(2):143–164, 2018.

2 Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014.

3 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.*, 87:119–139, 2017.

4 Andreas Björklund, Petteri Kaski, and Lukasz Kowalik. Fast witness extraction using a decision oracle. In *Proceedings of the 22nd Annual European Symposium on Algorithms (ESA '14)*, volume 8737 of *LNCS*, pages 149–160. Springer, 2014.

5 Laurent Bulteau, Falk Hüffner, Christian Komusiewicz, and Rolf Niedermeier. Multivariate algorithmics for NP-hard string problems. *Bulletin of the EATCS*, 114, 2014.

6 Peter Burcsi, Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. Algorithms for jumbled pattern matching in strings. *Int. J. Found. Comput. Sci.*, 23(2):357–374, 2012.

7 Anne Condon, Ján Manuch, and Chris Thachuk. Complexity of a collision-aware string partition problem and its relation to oligo design for gene synthesis. In *Proceedings of the 14th International Computing and Combinatorics Conference (COCOON '08)*, volume 5092 of *LNCS*, pages 265–275. Springer, 2008.

8 Anne Condon, Ján Manuch, and Chris Thachuk. The complexity of string partitioning. *J. Discrete Algorithms*, 32:24–43, 2015.

9 Maxime Crochemore and Wojciech Rytter. *Jewels of stringology.* World Scientific, 2002.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.

11 Henning Fernau, Florin Manea, Robert Mercas, and Markus L. Schmid. Pattern matching with variables: Fast algorithms and new hardness results. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS '15)*, volume 30 of *LIPIcs*, pages 302–315. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

12 Klaus Jansen, Felix Land, and Kati Land. Bounding the running time of algorithms for scheduling and packing problems. *SIAM J. Discrete Math.*, 30(1):343–366, 2016.

13 Christian Komusiewicz, Mateus de Oliveira Oliveira, and Meirav Zehavi. Revisiting the parameterized complexity of maximum-duo preservation string mapping. In *Proceedings of the 28th Annual Symposium on Combinatorial Pattern Matching (CPM '17)*, volume 78 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

14 Lukasz Kowalik and Juho Lauri. On finding rainbow and colorful paths. *Theor. Comput. Sci.*, 628:110–114, 2016.

15 Radu Stefan Mincu and Alexandru Popa. The maximum equality-free string factorization problem: Gaps vs. no gaps. In *Proceedings of the 45th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '20)*, volume 12011 of *LNCS*, pages 531–543. Springer, 2020.

16 Markus L. Schmid. Computing equality-free and repetitive string factorisations. *Theor. Comput. Sci.*, 618:42–51, 2016.