On Matching Generalised Repetitive Patterns

Joel D. Day[1], Pamela Fleischmann[1], Florin Manea[1], Dirk Nowotka[1], and Markus L. Schmid[2]

[1] Kiel University, Germany. {jda,fpa,flm,dn}@informatik.uni-kiel.de
[2] Trier University, Germany. mschmid@uni-trier.de

Abstract. A pattern is a string with terminals and variables (which can be uniformly replaced by terminal words). Given a class $\mathcal{C}$ of patterns (with variables), we say a pattern α is a $\mathcal{C}$-(pseudo-)repetition if its skeleton – the result of removing all terminal symbols to leave only the variables – is a (pseudo-)repetition of a pattern from $\mathcal{C}$. We introduce a large class of patterns which generalises several known classes such as the k-local and bounded scope coincidence degree patterns, and show that for this class, $\mathcal{C}$-(pseudo-)repetitions can be matched in polynomial time. We also show that for most classes $\mathcal{C}$, the class of $\mathcal{C}$-(pseudo-)repetitions does not have bounded treewidth. Finally, we show that if the notion of repetition is relaxed, so that in each occurrence the variables may occur in a different order, the matching problem is NP-complete, even in severely restricted cases.

Keywords: Pattern Matching with Variables · Repetitions

1 Introduction

A *pattern* is a word α consisting of symbols from a terminal alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots\}$, and a set of variables $X = \{x_1, x_2, \ldots\}$. A word $w \in \Sigma^*$ *matches* the pattern α if there exists a coherent substitution of the variables (i.e., each occurrence of a variable is replaced by the same string) under which α becomes equal to w. For example, the word $w = \mathtt{abababcbabaacac}$ matches the pattern $\alpha_1 = x_1\mathtt{b}x_2\mathtt{b}x_1x_2\mathtt{ac}$, as witnessed by the substitution $x_1 \to \mathtt{aba}$, $x_2 \to \mathtt{ac}$. On the other hand, as can be easily verified, w does not match $\alpha_2 = \mathtt{a}x_1\mathtt{c}x_1$. Thus, a word matches a pattern if it adheres to the underlying structure described by the pattern. The problem of deciding whether a given word matches a pattern is called the *matching problem*.[3]

Patterns with variables and the matching problem appear in various areas of theoretical computer science, such as combinatorics on words (word equations [15, 20], unavoidable patterns [17]), pattern matching (generalized function matching [1, 22]), language theory (pattern languages [2]), learning theory (inductive inference [2, 21, 23, 6], PAC-learning [16]), database theory (extended conjunctive regular path queries [3]), as well as in practical applications, e.g., extended regular expressions with backreferences [4, 12, 11], used in programming languages like Perl, Java, Python, etc.

[3] The matching problem is the same as the membership problem for *pattern languages*.

One important feature of patterns is that they can model very naturally repetitive structures, which appear in a wide range of applications from, e.g., music to biology. For instance, structures appearing in genetic data, such as tandem or inverted repeats, or hairpin structures, can be modelled by generalised patterns with variables and their reversals. As such, pseudo-repetitions (generalised repetitions of a word w and its reversal w^R) were studied from both combinatorial and algorithmic points of view, in the framework of matching patterns with variables (see, e. g., [14, 13, 18] and the references therein).

Unfortunately, this expressive power comes at a price: the matching problem is NP-complete in the general case [2]. This has lead to a thorough analysis of the classical and parameterised complexity of the matching problem (see [9, 10, 7]), which has shown that, in terms of simple numerical parameters, many severely restricted variants remain computationally hard. On the other hand, more complex structural parameters, e. g., the scope-coincidence degree [24] and k-locality [5], have been introduced, which yield classes of patterns which can be matched in polynomial time. For some stronger restrictions, like regular and non-cross patterns, there are also quite efficient matching algorithms (see [7]). All known efficiently matchable classes have a unifying theory: they can all be shown to have bounded treewidth (which is defined via a natural graph representation of patterns). Due to a meta-theorem of [24], if a class of patterns has this property, then the matching problem can be solved in polynomial time for this class.

It is interesting to note that the general treewidth-based framework of polynomial time matching of patterns does not seem to cover a very simple and natural aspect: repetitions of the same pattern. More precisely, if $\mathcal{C}$ is one of the known efficiently matchable classes of patterns, then a repetition α^k for some $\alpha \in \mathcal{C}$ is usually not in $\mathcal{C}$ anymore. In fact, we show here that even for patterns α with bounded and very low treewidth, the treewidth of repetitions α^k can be unbounded (see Theorem 2). Nevertheless, it is a very simple observation that if we can match patterns from a class $\mathcal{C}$ in polynomial time, then we can also match repetitions of these patterns in polynomial time: if we wish to check whether α^k matches a word w, then we can firstly check whether $w = v^k$ for some word v, and then check whether α matches v.

Taking a closer look at this phenomenon, it can be observed that most parameters that lead to efficiently matchable classes, e. g., the scope coincidence degree, are defined independently from the terminal symbols, i. e., via the word obtained after removing all terminals, which shall be called *skeleton* in the following (e. g., the skeleton of α_1 from above is $x_1x_2x_1x_2$). As a result, it is possible that a pattern, that is *not* a repetition of any $\alpha \in \mathcal{C}$, has nevertheless a skeleton that is a repetition of a skeleton from $\mathcal{C}$. For example, $\mathtt{a}x_1(x_2)^3x_3\mathtt{b}x_3x_1(x_2)^2\mathtt{b}x_2\mathtt{a}x_3^2$ is not a repetition of a non-cross pattern, but its skeleton $(x_1(x_2)^3(x_3)^2)^2$ is. We are able to show that, for some important classes $\mathcal{C}$ of patterns, the polynomial time solvability of the matching problem does not only extend from $\mathcal{C}$ to exact repetitions, but also to such skeleton-repetitions, called *$\mathcal{C}$-repetitions* here.

More precisely, motivated both by the previous work on finding large classes of patterns that can be matched in polynomial time and by the interest in repet-

itive patterns and their variations, we consider the following natural question: for which classes $\mathcal{C}$ of patterns, that can be matched in polynomial time, does the class of $\mathcal{C}$-repetitions have a polynomial-time matching problem? For our first main result, we define the parameterised class, $\mathrm{Mem}(k,p)$, where $k,p \in \mathbb{N}_0$, which extends both the classes of k-local patterns and patterns with scope coincidence degree at most p, and therefore also contains most of the other known efficiently matchable classes. Intuitively speaking, the k-local patterns can be matched by assigning the variables in a specific order such that at any point, no more than k separate factors of the pattern must be stored, while in the case of scope coincidence degree, the pattern is matched left-to-right, with only the values of at most p already-matched variables stored in case they appear later in the pattern. In the case of matching $\mathrm{Mem}(k,p)$ patterns, we can combine these strategies and store both at most k previously-matched factors, as well as the values of up to p variables which may be encountered again later in the process. Using the resulting algorithm as a starting point, we show that also $\mathrm{Mem}(k,p)$-repetitions can be matched in polynomial time. Moreover, the same result holds in the case of $\mathrm{Mem}(k,p)$-pseudo-repetitions, where $\mathrm{Mem}(k,p)$-repetitions are extended to allow also reversals (so $x_1x_2 \cdot x_2x_1 \cdot x_2x_1$ is a pesudo-repetition of x_1x_2). We also expect that our approach can be adapted to "plug-in" other dynamic programming based matching algorithms for other classes.

As already briefly mentioned above, we also point out the following fact: for the class REG of regular patterns – for which each variable may occur only once – the class of REG-repetitions does not have bounded treewidth and is thus not covered by the meta-theorem of [24]. Since the regular patterns are arguably the simplest class allowing an unbounded number of variables (note that patterns with a constant number of variables can trivially be matched in polynomial-time), this also holds for all the other classes mentioned previously, and in particular for $\mathrm{Mem}(k,p)$. To the knowledge of the authors, this is the first example of an efficiently matchable class of patterns, that does not have bounded treewidth, which is of theoretical interest with respect to the results from [24]. We are also able to show that for REG-repetitions, there is a more direct and efficient algorithm that solves the matching problem in time $O(m|w|)$, where w is the word to be matched and m is the number of variables.

Finally, we show that if the notion of a repetition is relaxed further than just allowing, e.g., reversals, by considering a setting where the order in which the variables appear is no longer constrained at all (such repetitions are often called *abelian repetitions* in the literature), then the matching problem is again NP-complete. Furthermore, this holds even in the minimal case that the number of repetitions is restricted to two, and that the pattern which is repeated is regular.

Due to space constraints, most proofs are omitted.

2 Basic Definitions

For detailed definitions regarding combinatorics on words we refer to [17]. For $n,i,j \in \mathbb{N}_0$ with $i \leq j$, let $[n] = \{1,\ldots,n\}$ and $[i,j] = \{i,i+1,\ldots,j-1,j\}$.

In this paper, $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots\}$ denotes a finite alphabet of *terminal symbols* and $X = \{x_1, x_2, \ldots\}$ a potentially infinite alphabet of *variables*. We assume $\Sigma \cap X = \emptyset$. Words in $(X \cup \Sigma)^*$ are *patterns*, while words in Σ^* are *terminal words* (usually just words). The *empty word* is denoted by ε and the *length* of a word w by $|w|$. Given a pattern α, let $\mathsf{var}(\alpha)$ be the smallest set $Y \subseteq X$ such that $\alpha \in (\Sigma \cup Y)^*$. Given a word $w = \mathtt{a}_1\mathtt{a}_2 \ldots \mathtt{a}_n$, the reversal w^R of w is $\mathtt{a}_n\mathtt{a}_{n-1} \ldots \mathtt{a}_1$. For $w \in \Sigma^*$ and each $i, j \in [|w|]$ with $i \leq j$, let $w[i..j] = w[i] \cdots w[j]$, where $w[k]$ represents the k^{th} letter of w for $k \in [|w|]$. Each word $w[i..j]$ is a *factor* of w. If $0 < |w[i..j]| < |w|$ then $w[i..j]$ is a *proper factor* of w.

A *substitution* (*for* α) is a mapping $h : \mathsf{var}(\alpha) \to \Sigma^+$. For every $x \in \mathsf{var}(\alpha)$, we say that x *is substituted by* $h(x)$. The word obtained by substituting every occurrence of a variable x in α by $h(x)$ and leaving the terminals unchanged is denoted by $h(\alpha)$. For instance, $h(x_1\mathtt{a}x_2\mathtt{b}x_2) = \mathtt{bacabca}$, where $h(x_1) = \mathtt{b}$, $h(x_2) = \mathtt{ca}$. The set $L(\alpha) = \{h(\alpha) \mid h \text{ is a substitution}\}$ is the *pattern language* of α. The *matching problem* is to decide for a given pattern α and word w, whether there exists a substitution h with $h(\alpha) = w$.[4]

A pattern α is *regular* if each variable $x \in X$ occurs at most once. Given a pattern α and $y \in \mathsf{var}(\alpha)$, the *scope of* y *in* α is defined by $\mathrm{sc}_\alpha(y) = [i, j]$, where i is the leftmost and j the rightmost occurrence of y in α. The scopes of some variables $y_1, y_2, \ldots, y_k \in \mathsf{var}(\alpha)$ *coincide in* α if $\bigcap_{1 \leq i \leq k} \mathrm{sc}_\alpha(y_i) \neq \emptyset$. We denote the *scope coincidence degree* (scd for short) of α by $\mathrm{scd}(\alpha)$, which is the maximum number of variables in α such that their scopes coincide. For example, $\mathrm{scd}(x_1x_2x_1x_2x_3x_1x_2x_3) = 3$ and $\mathrm{scd}(x_1x_2x_1x_2x_3x_2x_3x_3) = 2$. The class of *non-cross* patterns (see [25]) coincides exactly with the patterns with scope coincidence degree of 1.

The *skeleton* of $\beta \in (X \cup \Sigma)^*$ is the (unique) pattern $\mathsf{skel}(\beta) = y_1 \ldots y_n$ with $y_i \in X$ for $i \in [n], n \in \mathbb{N}$, such that there exist words $a_0, \ldots, a_n \in \Sigma^*$ with $\beta = a_0y_1a_1 \ldots a_{n-1}y_na_n$. For example, $\mathsf{skel}(\mathtt{aa}x\mathtt{c}y\mathtt{ba}z\mathtt{a}y\mathtt{a}y) = xxyzyy$. A class $\mathcal{C}$ of patterns is called a skel-class of patterns if $\alpha \in \mathcal{C} \Leftrightarrow \mathsf{skel}(\alpha) \in \mathcal{C}$.

Next, we recall the notion of a *marking sequence*. For each variable $x \in X$, $\overline{x}$ is its *marked version*, *marking* x means to substitute *every* occurrence of x with ist marked version. Let $\overline{X} = \{\overline{x} \mid x \in X\}$ be the set of *marked variables* (with $\overline{X} \cap X = \emptyset$). For the skeleton α of a pattern $\beta \in (X \cup \Sigma)^*$, a *marking sequence* of the variables occurring in β, is an ordering $x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(m)}$ of $\mathsf{var}(\beta) = \{x_1, x_2, \ldots, x_m\}$, where σ is a permutation of $[m]$. A variable x is called *marked at point* $k \in \mathbb{N}$ (both in β and α) if $x = x_{\sigma(i)}$ for some $i \leq k$. Similarly, a position of α is marked at point k if it corresponds to a variable that is marked at point k. A factor of a string α containing marked positions is a *marked block* if it consists of one or more marked positions and is maximal in the sense that it is not contained within another such factor. According to [5], for a pattern β with $\mathsf{var}(\beta) = \{x_1, \ldots, x_m\}$ and $\alpha = \mathsf{skel}(\beta)$, we say that β is k-local, if there

[4] The difference of whether variables can be substituted by ε may seem negligible but is crucial for some aspects of pattern languages (*erasing* versus *non-erasing* pattern languages). Since here we are concerned with the matching problem, we only point out whether results also apply to the erasing case, or how they can be extended.

exists a marking sequence $x_{\sigma(1)}, \ldots, x_{\sigma(m)}$, such that, for $1 \leq i \leq m$, the string obtained from α in i marking steps contains at most k marked blocks.

For a pattern α, an *α-structure* is an undirected graph with two types of edges, i. e., $G_\alpha = (V, P, E)$, where $V = \{1, 2, \ldots, |\alpha|\}$ is the set of vertices, $P = \{\{i, i+1\} \mid i \in [|\alpha| - 1]\}$ connects all vertices into a path and E is an edge-relation that only connects vertices that correspond to different occurrences of the same variable, such that, for each variable, the set of its occurrences forms a connected component of G with respect to edge-relation E. If a class $\mathcal{C}$ of patterns has α-structures with bounded treewidth, then the matching problem for $\mathcal{C}$ can be solved in polynomial-time (see [24] for details).

The computational model we use in this work is the standard unit-cost RAM with logarithmic word size.

3 Mem(k, p)-Patterns

In a way similar to the definition of k-local patterns from the previous section, we use the idea of a marking sequence to introduce the class of Mem(k, p)-patterns.

Essentially, the definition of Mem(k, p)-patterns is motivated by the following idea. Generally, algorithms checking whether a pattern with variables matches a word assign values to the variables in a certain order (e.g., from left to right, by a specified marking sequence, etc.). However, we may follow two strategies. On the one hand, we can store the factors that contain all the occurrences of the variables for which we assigned values (and forget the individual assignment) and their matching factors from the word. This strategy is used, for instance, to match k-local patterns. On the other hand, we can store the individual images of the assigned variables, and try to match, e.g., an increasingly large prefix of the pattern to a prefix of the word, using these images. In this strategy, used, e.g., to match patterns with bounded scope coincidence degree, we have to store the image of a variable until we reach its latest occurrence. In both strategies, if the amount of data we have to store is constant, we get a polynomial time matching algorithm. In order to make this happen for certain classes, structural restrictions on the patterns were used.

Here we combine these two competing ideas. We introduce a class of patterns where, in order to match them to a word, we again assign values to the variables of the pattern in an order given by a marking sequence. However, in this case, we want to store, after each new assignment, a match between $k' \leq k$ factors of the pattern (like in the first approach mentioned above), but, we are more flexible, and also store the assignment of at most p of the variables (like in the second case described above) that are allowed to reoccur also outside the k' factors we matched. This new class (parametrised by the integers k and p) can, once more, be defined by structural restrictions on the pattern.

Definition 1. *Let $k \geq 1$ and $p \geq 0$ be two integers. Let β be a pattern with $\mathsf{var}(\beta) = \{x_1, \ldots, x_m\}$ and $\alpha = \mathsf{skel}(\beta)$. We say that β is a* Mem(k, p)*-pattern (or, alternatively, that it belongs to the class of* Mem(k, p) *patterns) if there exists*

a marking sequence $x_{\sigma(1)}, \ldots, x_{\sigma(m)}$ *of the variables from* $\mathsf{var}(\beta)$ *such that the following holds. For* $1 \leq i \leq m$*, after marking* $x_{\sigma(1)}, \ldots, x_{\sigma(i)}$ *in* α*, we have that the set of marked positions of* α *can be partitioned into:*

1. $k_i \leq k$ *disjoint intervals (of marked positions) that define exactly* k_i *marked blocks of* α*:*
$$\alpha[a_{i,1}..b_{i,1}]; \alpha[a_{i,2}..b_{i,2}]; \ldots; \alpha[a_{i,k_i}..b_{i,k_i}].$$
2. *the remaining marked positions, which correspond to occurrences of at most* p *variables from* $\{x_{\sigma(1)}, \ldots, x_{\sigma(i)}\}$.

Moreover, for all $i \geq 2$*, and each* $j \leq k_{i-1}$ *we have that there exists* ℓ *such that* $a_{i,\ell} \leq a_{i-1,j} \leq b_{i-1,j} \leq b_{i,\ell}$ *(i.e., the* k_{i-1} *marked blocks from step* $i-1$ *are contained in the* k_i *marked blocks at step* i*).*

In other words, a pattern β (and its skeleton α) is in $\mathrm{Mem}(k,p)$ if there exists a marking sequence of the variables occurring in α such that at each step of the marking we can partition the marked positions into $k_i \leq k$ marked blocks of α and a set of other separate marked positions, which, however, correspond to occurrences of at most p variables. Moreover, the $\leq k$ marked blocks from step i should extend the respective blocks from step $i-1$ (i.e., a marked position that was in one of the k_{i-1} marked blocks identified at step $i-1$ will still be in one of the k_i marked blocks from step i).

Example 1. Consider a pattern β with skeleton
$$\alpha = (x_1x_2x_1x_3x_1x_4 \ldots x_1x_n)^2 x_n x_{n-1} \ldots x_2x_1x_2x_3 \ldots x_n.$$
The variables of this pattern could be marked as follows. First mark x_1 and treat the rightmost occurrence of x_1 as one of the marked blocks, while the other occurrences are not considered as marked blocks, but x_1 is one of the marked variables (with many occurrence) we store additionally. This already shows that our pattern is at least $\mathrm{Mem}(1,1)$. Then we mark x_2, which extends the marked block on the right side, but in the left part, we now consider the two marked factors $x_1x_2x_1$ as marked blocks, while the remaining isolated marked occurrences of x_1 are still considered as occurrences of marked variables stored additionally to the marked blocks. This places the pattern in $\mathrm{Mem}(3,1)$ (3 marked blocks and 1 additional variable). We repeat this and mark $x_3, x_4, \ldots, x_n$ in this order, and we always consider that the newly marked occurrences of variables on the left part are joined to the existing blocks by some of the already marked occurrences of x_1 that were not previously in any marked block. In particular, the isolated marked occurrences of x_1 that were counted before as the single marked variable we store additionally (thus, defining the second component of $\mathrm{Mem}(k,p)$) change their role, and become part of the marked blocks (counted in the the first component of $\mathrm{Mem}(k,p)$). In conclusion, α and the pattern β with $\alpha = \mathsf{skel}(\beta)$ are in $\mathrm{Mem}(3,1)$.

It is immediate that $\mathrm{Mem}(k,p) \subseteq \mathrm{Mem}(k',p')$ if $k \leq k'$ and $p \leq p'$. Also, $\mathrm{Mem}(k,p)$ contains the class of regular patterns for all k and p. The next example shows that even $\mathrm{Mem}(1,1)$ contains rather complex patterns.

Example 2. $(x_1x_2)^n x_1x_2\cdots x_{n-1}x_nx_{n-1}\cdots x_2x_1 \in \mathrm{Mem}(1,1)$.

As already mentioned, $\mathrm{Mem}(k,p)$ patterns combine the definitions of k-local patterns and patterns with scope coincidence degree bounded by p (and, in fact, provide a unified view of the known classes of patterns that can be matched by dynamic programming – hence, the name Mem from memoization). We explain this in more details. Consider, in the following, that α is the skeleton of a pattern. If α is k-local, then we have a marking sequence that defines at each step at most k marked blocks of α. If α is a pattern with scope coincidence degree bounded by p we can mark the variables of α from left to right, and, at each step, we have a completely marked prefix and several other separate occurrences of at most p distinct marked variables. In the case when α is an $\mathrm{Mem}(k,p)$ pattern, we again have a marking sequence that defines at each step k marked blocks of α; however, different from the case of k-local patterns, we might have, at each marking step, some other marked occurrences of variables outside the k marked blocks. But, as for patterns with bounded scope coincidence degree, the number of distinct marked variables occurring outside the k blocks is at most p (although, the number of *marked positions* outside of the marked blocks is not necessarily bounded by p or k, since the additional marked variables can occur arbitrarily many times outside of marked blocks). Example 2 shows that $\mathrm{Mem}(1,1)$ contains patterns of length $4n-1$ which are $(n+1)$-local and have scope coincidence degree n. Obviously, arbitrary patterns which are k-local but not $k-1$ local are in $\mathrm{Mem}(k,0)$ but not in $\mathrm{Mem}(k-1,0)$, while the pattern $(x_1x_2\ldots x_p)^n$ is not in $\mathrm{Mem}(1,p-2)$.

It is not hard to show that one can decide whether a pattern is in $\mathrm{Mem}(k,p)$ in polynomial time (where the degree of the polynomial depends on k and p).

Proposition 1. *Given a pattern β, we can decide in $O(|\beta|^{3k+p+4})$ time whether $\beta \in \mathrm{Mem}(k,p)$. Moreover, if β is in $\mathrm{Mem}(k,p)$ then we can also produce a marking sequence for it, as well as, for each step i of this marking sequence, the $k_i \leq k$ maximal factors marked obtained after performing the marking in step i and the $p_i \leq p$ marked variables that occur outside these k marked factors.*

Basically, if β is in $\mathrm{Mem}(k,p)$, we can construct in $O(|\beta|^{3k+p+4})$ time a marking sequence $x_{d_1},\ldots,x_{d_m}$ of the variables of $\alpha = \mathsf{skel}(\beta)$ and a corresponding sequence of tuples $a_s = (i_1^s, j_1^s, \ldots, i_k^s, j_k^s, \ell_1^s, \ldots, \ell_p^s)$, encoding the marked positions of α after $x_{d_1},\ldots,x_{d_s}$ were marked. The tuple $a_s = (i_1^s, j_1^s, \ldots, i_k^s, j_k^s, \ell_1^s, \ldots, \ell_p^s)$ encodes the information that, after s marking steps, the blocks $\alpha[i_e^s..j_e^s]$ are marked, and on all the other marked positions of α we have only variables from $x_{\ell_1^s},\ldots,x_{\ell_p^s}$. If we actually need to store only $k' < k$ marked blocks after the first s marking steps, then $i_e^s = j_e^s = |\alpha|+1$ for $e > k'$; similarly, if we need to store only $p' < p$ additional marked variables, then $\ell_h^s = m+1$ for $s > p'$.

Once we can obtain the information described above for the skeleton of a $\mathrm{Mem}(k,p)$-pattern β, we are able to compute the tuples $b_s = (l_1^s, r_1^s, \ldots, l_k^s, r_k^s, \ell_1^s, \ldots, \ell_p^s)$, for $s \in [m]$, encoding the marked symbols of β after s marking steps. We just mark the variables corresponding to the marked blocks of α, and the terminals occurring between the variables of such a block. Then, given a word

w, we can use dynamic programming to track all the possible ways the k marked blocks obtained in the pattern after s marking steps may match factors of w and the p additional marked variables can be coherently assigned factors of w, and, ultimately, when $s = m$, check whether β matches w.

Proposition 2. *Given a pattern $\beta \in \mathrm{Mem}(k, p)$ and a word w, of length n, we can decide in $O(n^{3k+2p+5})$ time whether β matches w.*

4 Generalized Repetitions of Patterns

We first introduce the notion of $\mathcal{C}$-repetition, where $\mathcal{C}$ is a skel-class of patterns, like, for instance, the class of regular patterns, non-cross patterns, patterns with bounded scope coincidence degree, k-local patterns, or $\mathrm{Mem}(k, p)$ patterns.

Definition 2. *Let $\beta \in (X \cup \Sigma)^*$ and $\mathcal{C}$ a class of patterns. We call β a $\mathcal{C}$-repetition if there exist a positive integer $r \geq 2$, patterns $\beta_1, \ldots, \beta_r \in \mathcal{C}$, with $\beta_i \in (X \cup \Sigma)^*$ for all $i \in [r]$, and $\alpha \in X^*$, such that $\beta = \beta_1 \cdots \beta_r$ and $\mathsf{skel}(\beta_i) = \alpha$ for all $i \in [r]$. The pattern α is called a $\mathcal{C}$-root-skeleton of β.*

It is not hard to see that, in the definition above, we have that if $\mathcal{C}$ is a skel-class of patterns, then α is in $\mathcal{C}$ as well. In general, this result might not hold (e.g., when $\mathcal{C}$ is defined by a property regarding the terminals of the patterns).

It is well known (see, e.g., [17]) that if a word w fulfils $w = u^k$ and $w = v^\ell$ for u, v words and $k, \ell \geq 2$, then there exists a word t, called the root of w, $|t| \leq \min\{|u|, |v|\}$ such that $u = t^p$ and $v = t^r$. This does not hold when we talk about $\mathcal{C}$-repetitions and their $\mathcal{C}$-root-skeletons. Indeed, if $\mathcal{C}$ is the class of patterns that do not contain single occurrences of a variable (i.e., patterns β with $|\beta|_x \geq 2$ for all $x \in \mathsf{var}(\beta)$), then we have two $\mathcal{C}$-root-skeletons of $(x_1x_2)^6$, namely $(x_1x_2)^2$ and $(x_1x_2)^3$; however, x_1x_2, which is a root (as a word, in the classical combinatorics on words sense) of each of the $\mathcal{C}$-root-skeletons, is not a $\mathcal{C}$-root-skeleton of $(x_1x_2)^6$. However, it is rather easy to show that all $\mathcal{C}$-root-skeletons are powers of a common root (when interpreted as words).

For simplicity, we call the shortest $\mathcal{C}$-root-skeleton of a $\mathcal{C}$-repetition β *the $\mathcal{C}$-root-skeleton* of a $\mathcal{C}$-repetition β, and we denote it by $\mathsf{rskel}_{\mathcal{C}}(\beta)$. The $\mathcal{C}$-exponent of β is defined as $\mathsf{exp}_{\mathcal{C}}(\beta) = \frac{|\,\mathsf{skel}(\beta)|}{|\,\mathsf{rskel}_{\mathcal{C}}(\beta)|}$.

Definition 2 can be extended to $\mathcal{C}$-pseudo-repetitions. For simplicity, we will address directly the case when $\mathcal{C}$ is a skel-class of patterns.

Definition 3. *Let $\beta \in (X \cup \Sigma)^*$ and $\mathcal{C}$ a skel-class of patterns. We call β a $\mathcal{C}$-pseudo-repetition if there exist a positive integer $r \geq 2$, patterns $\beta_1, \ldots, \beta_r$, with $\beta_i \in (X \cup \Sigma)^*$ for all $i \in [r]$, and $\alpha \in X^*$ a pattern of class $\mathcal{C}$, such that $\beta = \beta_1 \cdots \beta_r$ and $\mathsf{skel}(\beta_i) = \alpha$ or $\mathsf{skel}(\beta_i) = \alpha^R$ for all $i \in [r]$. The pattern α is called a $\mathcal{C}$-pseudo-root-skeleton of β, and the shortest $\mathcal{C}$-pseudo-root-skeleton of β is called* the $\mathcal{C}$-pseudo-root-skeleton *of β, denoted $\mathsf{prskel}_{\mathcal{C}}(\beta)$. The $\mathcal{C}$-exponent of the β is defined as $\mathsf{exp}_{\mathcal{C}}(\beta) = \frac{|\,\mathsf{skel}(\beta)|}{|\,\mathsf{prskel}_{\mathcal{C}}(\beta)|}$.*

In the following, we focus on a series of algorithmic results regarding the notions introduced above. Recall that the membership problem for a class $\mathcal{C}$ of patterns is the decision problem asking whether a given pattern α is in $\mathcal{C}$ or not. The membership problem can be decided in polynomial time for each of the class of regular patterns, the class of non-cross patterns, the class of patterns with bounded scope coincidence degree, the class of k-local patterns, or the class of $\mathrm{Mem}(k, p)$ patterns. Our first result is immediate.

Proposition 3. *Let $\mathcal{C}$ be a skel-class of patterns for which the membership problem can be decided in $O(f(n))$ time, for a polynomial f. Given a pattern β we can decide in $O(|\beta|^2 f(|\beta|))$ whether it is a $\mathcal{C}$-repetition (or, alternatively, a $\mathcal{C}$-pseudo-repetition). We can compute, in the same time, $\mathsf{rskel}_{\mathcal{C}}(\beta)$ (resp., $\mathsf{prskel}(\beta)$).*

In the following we address the matching problem for repetitions. More precisely, we show that $\mathrm{Mem}(k, p)$-repetitions can be matched in polynomial time, if the parameters k and p are upper bounded by constants.

Theorem 1. *Given β a $\mathrm{Mem}(k, p)$-repetition and a word w, of length n, we can decide whether β matches w in $O(n^{3k+2p+5})$ time.*

Firstly, we compute the root-skeleton α of β, the exponent r, and partition $\beta = \beta_1 \dots \beta_r$, where, for $i \in [r]$, α is β_i's skeleton. By choosing the factors β_i such that each ends as far to the right as possible, and because the total length of the variables' images should be the same in all the factors β_i, we can compute by length arguments the corresponding decomposition of $w = w_1 \dots w_r$ such that if w matches β, then, for each i, w_i must match β_i. By Proposition 1, we get a marking sequence $x_{d_1}, \dots, x_{d_m}$ for β_1, and note that this is also a valid marking sequence for each β_h, with $h \geq 2$. Then, we try to match w_i and β_i, for all $i \in [r]$, simultaneously and coherently. However, keeping track in a data structure how each of the k marked blocks of β_i match factors of the corresponding word w_i, for all $i \in [r]$, would lead to an algorithm exponential in r. Instead, we notice that if we know the marked block of β_1 after s markings steps, we can construct the k marked factors of each of β_h after s marking steps, for $h \geq 2$, as all these patterns have the same skeletons. Moreover, if we discovered that one of the blocks of β_1 that are marked after s steps, say $\beta_1[l..r]$, should match $w_1[i..j]$, we can compute, for all $h \geq 2$, again by length arguments (as $\mathsf{skel}(\beta_h) = \mathsf{skel}(\beta_1)$ and their variables are assigned coherently), the factor $w_h[i_h..j_h]$ which should match $\beta_h[l_h..r_h]$, the marked block of β_h that corresponds to $\beta_1[l..r]$.

Hence, if $b_s = (l_1^s, r_1^s, \dots, l_k^s, r_k^s, \ell_1^s, \dots, \ell_p^s)$ encodes the factors $\beta_1[l_e^s..r_e^s]$ of β_1 that are marked after s marking steps, as well as the additional p marked variables, we compute and store the tuples $(s, i_1, j_1, \dots, i_{k+p}, j_{k+p})$ such that $\beta_1[l_e^s..r_e^s]$ matches $w_1[i_e..j_e]$, for $e \in [k]$, and the x_{ℓ_f} is assigned the value $w_1[i_{k+f}..j_{k+f}]$, for $f \in [p]$. Moreover, we only store from these tuples those that induced a valid matching for the marked blocks of β_h, for $h \geq 2$, as explained above. So, a matching between a tuple b_s and a corresponding tuple $(s, i_1, j_1, \dots, i_{k+p}, j_{k+p})$ encodes a matching between some factors of β_1 and factors of w_1 which can be extended coherently to a matching between the factors

of β_h and factors of w_h. If we have all these matching tuples for some s, we try to extend them, using dynamic programming, to similar matchings for $s+1$. To do this, when considering a tuple $t = (s, i_1, j_1, \ldots, i_{k+p}, j_{k+p})$, we try to assign values to $x_{d_{s+1}}$ that allow for a matching of the marked blocks of β_1 encoded by $b_{s+1} = (l_1^{s+1}, r_1^{s+1}, \ldots, l_k^{s+1}, r_k^{s+1}, \ell_1^{s+1}, \ldots, \ell_p^{s+1})$ to factors of w_1 which extend the factors encoded in t, and also check whether similar matchings hold for all $h \geq 2$ (i.e., the implicit matching that follows from the matching of the factors encoded by b_s to those encoded by t are also extended in a valid and coherent way). The algorithm ends after we found all the matchings after m marking steps, and decides that β matches w if and only if we were able to find a valid matching of β_1 to w_1, which induces a matching of β_h to w_h, for all $h \geq 2$.

Our approach can be easily extended for pseudo-repetitions.

Corollary 1. *Given β a* Mem(k,p)*-pseudo-repetition and a word w, of length n, we can decide whether β matches w in $O(n^{3k+2p+5})$ time.*

Remark 1. In the algorithm of Theorem 1, as well as in its variant from Corollary 1, when we assigned a value to a variable we made no assumption on whether it is the empty string or not. So, our algorithms work also for the erasing-case, i. e., where variables can be substituted by the empty word as well.

We conclude this section by showing a structural result.

Theorem 2. *Let $\mathcal{C}$ be a class of patterns that contains all* REG*-patterns. Then the class of $\mathcal{C}$-repetitions contains patterns with arbitrarily large treewidth.*

The results in [24] show that patterns with bounded treewidth are matchable in polynomial time. Theorems 1 and 2 lead to a series of simple and natural examples of classes of patterns that can be matched in polynomial time, although they do not have bounded treewidth.

5 REG-repetitions

The results in Theorems 1 and Corollary 1 show that repetitions with the root- or pseudo-root-skeleton from some efficiently matchable classes of patterns can also be matched in polynomial time. However, it is to be expected that in the case of some more restrictive classes of patterns more efficient algorithms can be obtained, by exploiting their particularities. Indeed this holds for the class of REG-repetitions and -pseudo-repetitons. It is worth noting that these algorithms are based on an efficient implementation of the general dynamic programming algorithm we used for matching Mem(k,p) patterns, which exploits the simpler structure of regular patterns and makes use of some string processing data-structures. Note that, due to the fact that matching a REG-repetition means matching several regular patterns simultaneously and coherently, we cannot rely on the usual greedy strategy used to match a single regular pattern (see, e.g., [7]).

Theorem 3. *Given a* REG*-(pseudo-)repetition β, with $|\mathsf{var}(\beta)| = m$, and a word w, with $|w| = n$, we can decide whether β matches w in $O(mn)$ time.*

The positive algorithmic results from the previous section show that patterns of the form $\beta_1\beta_2\dots\beta_r$ can be matched efficiently, provided that each β_i is efficiently matchable *and* all β_i are "identical" (i. e., identical with respect to their skeleton or reversed skeleton). In the following, we shall investigate whether the second condition can be relaxed without losing these beneficial algorithmic properties. A natural respective approach is to allow the β_i to differ in a more complicated way as that one is the reversal of the other, e. g., by only requiring one to be a permutation of the other. We can show the rather strong negative result that even for $r = 2$ and regular β_i, this leads to NP-hardness.

Theorem 4. *Deciding whether a pattern $\beta_1\beta_2$, with $\mathsf{var}(\beta_1) = \mathsf{var}(\beta_2)$ and $\beta_1, \beta_2 \in \mathrm{REG}$, matches a word w is NP-hard.*

To prove Theorem 4 the *perfect code problem* for 3-regular graphs will be reduced to the matching problem of the specific kind in this theorem (similar to reductions in [8, 19]). Let $G = (V, E)$ with $V = \{t_1, \dots, t_n\}$ be a 3-regular graph. To get a convenient access to the neighbours of a given $v \in V$ define for $r \in [4]$ the (not unique) mappings $\wp_r : [n] \to [n]$ where $\wp_r(i) = j$ indicates that the r^{th} neighbour of t_i is t_j (they are assumed to be arbitrary but fixed). Define for a given graph G a pattern matching instance: let $X = \{x_{i,j} \mid i, j \in [n]\} \cup \{y_i, y_i' \mid i \in [n]\}$ be the set of variables and $\Sigma = \{\mathtt{a}, \#, \star\}$ be the set of terminal symbols. For all $i \in [n]$ set $\alpha_i = x_{\wp_1(i),i} \dots x_{\wp_4(i),i}$, $\alpha_i' = y_i \# x_{i,\wp_1(i)} \dots x_{i,\wp_4(i)} \# y_i'$, $w_i = \mathtt{a}^5$, $w_i' = (\#\mathtt{a}^8)^2 \#(\mathtt{a}^4\#)^2$, $\beta_1 = \alpha_1 \star \dots \star \alpha_n \star y_1 \dots y_n y_1' \dots y_n' \star$, $\beta_2 = \alpha_1' \star \dots \star \alpha_n'$, and $v_1 = w_1 \star \dots \star w_n \star (\#\mathtt{a}^8)^{2n - \frac{n}{4}} (\mathtt{a}^4\#)^{n + \frac{n}{4}} \star$, $v_2 = w_1' \star \dots \star w_n'$. Finally set $\beta = \beta_1\beta_2$ and $v = v_1 v_2$. Notice that, in contrast to β, v contains only terminal symbols and hence (β, v) is a pattern matching instance.

By definition, β_1 and β_2 are regular and, since, for every $i, j \in [n]$, $x_{i,j}$ occurs in α_j if and only if it occurs in α_i', and all variables y_i, y_i', $i \in [n]$ occur in both β_1 and β_2, we get $\mathsf{var}(\beta_1) = \mathsf{var}(\beta_2)$, i. e., $\mathsf{skel}(\beta_1)$ is a permutation of $\mathsf{skel}(\beta_2)$. This, alongside the next result, allows us to reach the conclusion.

Lemma 1. *The graph G has a perfect code if and only if β matches to v.*

Remark 2. The reduction from above can easily be modified for erasing substitutions: Set $w_i = \mathtt{a}$ and $w_i' = \#\mathtt{a}^4\#\#$, and the factor matched against $y_1 \dots y_n y_1' \dots y_n'$ is then $(\#\mathtt{a}^4)^{n - \frac{n}{4}} \#^{\frac{n}{4}}$. The proof is analogous.

References

1. A. Amir and I. Nor. Generalized function matching. *Journal of Discrete Algorithms*, 5:514–523, 2007.
2. D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.
3. P. Barceló, L. Libkin, A. W. Lin, and P. T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems*, 37, 2012.

4. C. Câmpeanu, K. Salomaa, and S. Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14:1007–1018, 2003.
5. J. D. Day, P. Fleischmann, F. Manea, and D. Nowotka. Local patterns. In *Proc. 37th FSTTCS*, volume 93 of *LIPIcs*, pages 24:1–24:14, 2017.
6. T. Erlebach, P. Rossmanith, H. Stadtherr, A. Steger, and T. Zeugmann. Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science*, 261:119–156, 2001.
7. H. Fernau, F. Manea, R. Mercaş, and M. L. Schmid. Pattern matching with variables: Fast algorithms and new hardness results. In *Proc. 32nd STACS*, volume 30 of *LIPIcs*, pages 302–315, 2015.
8. H. Fernau and M. L. Schmid. Pattern matching with variables: A multivariate complexity analysis. In *Proc. 24th CPM*, volume 7922 of *Lecture Notes in Computer Science*, pages 83–94, 2013.
9. H. Fernau and M. L. Schmid. Pattern matching with variables: A multivariate complexity analysis. *Information and Computation*, 242:287–305, 2015.
10. H. Fernau, M. L. Schmid, and Y. Villanger. On the parameterised complexity of string morphism problems. *Theory Comput. Syst.*, 59(1):24–51, 2016.
11. D. D. Freydenberger. Extended regular expressions: Succinctness and decidability. *Theory of Computing Systems*, 53:159–193, 2013.
12. J. E. F. Friedl. *Mastering Regular Expressions*. O'Reilly, Sebastopol, CA, third edition, 2006.
13. P. Gawrychowski, T. I, S. Inenaga, D. Köppl, and F. Manea. Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes. *Theory Comput. Syst.*, 62(1):162–191, 2018.
14. P. Gawrychowski, F. Manea, and D. Nowotka. Testing generalised freeness of words. In *Proc. 31st STACS*, volume 25 of *LIPIcs*, pages 337–349, 2014.
15. J. Karhumäki, W. Plandowski, and F. Mignosi. The expressibility of languages and relations by word equations. *Journal of the ACM*, 47:483–505, 2000.
16. M. Kearns and L. Pitt. A polynomial-time algorithm for learning k-variable pattern languages from examples. In *Proc. 2nd COLT*, pages 57–71, 1989.
17. M. Lothaire. *Combinatorics on Words*. Cambridge University Press, 1997.
18. F. Manea, M. Müller, D. Nowotka, and S. Seki. The extended equation of Lyndon and Schützenberger. *J. Comput. Syst. Sci.*, 85:132–167, 2017.
19. F. Manea, D. Nowotka, and M. L. Schmid. On the solvability problem for restricted classes of word equations. In *Proc. 20th DLT*, volume 9840 of *Lecture Notes in Computer Science*, pages 306–318, 2016.
20. A. Mateescu and A. Salomaa. Finite degrees of ambiguity in pattern languages. *RAIRO Informatique Théoretique et Applications*, 28:233–253, 1994.
21. Y. K. Ng and T. Shinohara. Developments from enquiries into the learnability of the pattern languages from positive data. *Theoretical Computer Science*, 397:150–165, 2008.
22. S. Ordyniak and A. Popa. A parameterized study of maximum generalized pattern matching problems. In *Proc. 9th IPEC*, volume 8894 of *Lecture Notes in Computer Science*, pages 270–281, 2014.
23. D. Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*, 397:166–193, 2008.
24. D. Reidenbach and M. L. Schmid. Patterns with bounded treewidth. *Inf. Comput.*, 239:87–99, 2014.
25. T. Shinohara. Polynomial time inference of pattern languages and its application. In *Proc. 7th IBM MFCS*, pages 191–209, 1982.