



# Subsequences with Gap Constraints: Complexity Bounds for Matching and Analysis Problems

Joel D. Day  



Loughborough University, UK

Maria Kosche  

Computer Science Department, Universität Göttingen, Germany

Florin Manea  

Computer Science Department and CIDAS, Universität Göttingen, Germany

Markus L. Schmid  

Humboldt-Universität zu Berlin, Germany

---

## Abstract

We consider subsequences with gap constraints, i. e., length- $k$  subsequences  $p$  that can be embedded into a string  $w$  such that the induced gaps (i. e., the factors of  $w$  between the positions to which  $p$  is mapped to) satisfy given gap constraints  $gc = (C_1, C_2, \dots, C_{k-1})$ ; we call  $p$  a  $gc$ -subsequence of  $w$ . In the case where the gap constraints  $gc$  are defined by lower and upper length bounds  $C_i = (L_i^-, L_i^+) \in \mathbb{N}^2$  and/or regular languages  $C_i \in \text{REG}$ , we prove tight (conditional on the orthogonal vectors (OV) hypothesis) complexity bounds for checking whether a given  $p$  is a  $gc$ -subsequence of a string  $w$ . We also consider the whole set of all  $gc$ -subsequences of a string, and investigate the complexity of the universality, equivalence and containment problems for these sets of  $gc$ -subsequences.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Design and analysis of algorithms

**Keywords and phrases** String algorithms, subsequences with gap constraints, pattern matching, fine-grained complexity, conditional lower bounds, parameterised complexity

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2022.64

**Related Version** *Full Version*: <https://arxiv.org/abs/2206.13896>

**Funding** The work of the three authors with German affiliation was supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG). Maria Kosche's work was supported by the project with number 389613931. Florin Manea's work was supported by the project with number 466789228. Markus L. Schmid's work was supported by the project with number 416776735.

## 1 Introduction

For a string  $v = v_1v_2 \dots v_n$ , where each  $v_i$  is a single symbol from some alphabet  $\Sigma$ , any string  $u = v_{i_1}v_{i_2} \dots v_{i_k}$  with  $k \leq n$  and  $1 \leq i_1 \leq i_2 \leq \dots \leq i_k \leq n$  is called a *subsequence* (or *scattered factor* or *subword*) of  $v$  (denoted by  $u \preceq v$ ). This is formalised by the *embedding* from the positions of  $u$  to the positions of  $v$ , i. e., the increasing mapping  $e : \{1, 2, \dots, k\} \rightarrow \{1, 2, \dots, n\}$  with  $j \mapsto i_j$  (we use the notation  $u \preceq_e v$  to denote that  $u$  is a subsequence of  $v$  via embedding  $e$ ). For example, the string **a b a c b b a** has among its subsequences **a a a**, **a b c a**, **c b a**, and **a b a b b a**. With respect to **a a a**, there exists just one embedding, namely  $1 \mapsto 1, 2 \mapsto 3$ , and  $3 \mapsto 7$ , but there are two embeddings for **c b a**.

In this paper, we are interested in *subsequences with gap constraints* that can be embedded in such a way that the *gaps* of the embedding, i. e., the factors  $v_{e(i)+1}v_{e(i)+2} \dots v_{e(i+1)-1}$  between the images of the mapping, satisfy certain properties. We begin by discussing why the concept of classical subsequences (i. e., without gap constraints) is a central one in computer science, and then we will motivate and describe in detail our approach.



© Joel D. Day, Maria Kosche, Florin Manea, and Markus L. Schmid;  
licensed under Creative Commons License CC-BY 4.0

33rd International Symposium on Algorithms and Computation (ISAAC 2022).

Editors: Sang Won Bae and Heejin Park; Article No. 64; pp. 64:1–64:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The concept of subsequences is employed in many different areas of computer science: in formal languages and logics (e. g., piecewise testable languages [59, 60, 41, 42, 43], or subword order and downward closures [35, 46, 45, 66]), in combinatorics on words [55, 25, 48, 47, 57, 53, 56], for modelling concurrency [54, 58, 16], in database theory (especially *event stream processing* [5, 34, 67]). Moreover, many classical algorithmic problems are based on subsequences, e. g., longest common subsequence [8] or shortest common supersequence [52]. Note that the longest common subsequence problem, in particular, has recently regained substantial interest in the context of fine-grained complexity (see [14, 15, 1, 2]).

There are two main types of algorithmic problems for subsequences investigated in the literature. Firstly, *matching*: the problem to decide whether a string  $u$  is a subsequence of a string  $v$ , i. e., whether  $u \preceq v$  (the term matching is motivated by the point of view that  $u$  is a pattern that is to be matched with the string  $v$ ). Secondly, the *analysis problems* are concerned with the sets  $SubSeq(k, v)$  of all length- $k$  subsequences of a given string  $v$ . More precisely, for given string  $v \in \Sigma^+$  and integer  $k \in \mathbb{N}$ , we want to decide whether  $SubSeq(k, v) = \Sigma^k$  (*universality*), or, for an additional string  $v'$ , whether  $SubSeq(k, v) \subseteq SubSeq(k, v')$  (*containment*) or  $SubSeq(k, v) = SubSeq(k, v')$  (*equivalence*). For classical subsequences (as defined above), the matching problem is trivial, while the analysis problems are well-investigated and relatively well-understood. For instance, the equivalence problem was introduced by Imre Simon in his PhD thesis [59], and was intensely studied in the combinatorial pattern matching community (see [36, 32, 61, 63, 19, 24] and the references therein), before being optimally solved in 2021 [33]. In this work, we consider these problems with respect to an extended setting of subsequences, which we shall explain and motivate next.

**Motivation for Our Setting.** In the theoretical literature, problems on subsequences are usually considered in the setting where the embeddings (as witnesses for subsequences) can be arbitrary. This means that any subsequence  $u$  of string  $v$  is witnessed by a *canonical* embedding  $e$  that greedily maps each position  $i$  of  $u$  to the leftmost occurrence of symbol  $u_i$  in the suffix  $v_{e(i-1)+1} v_{e(i-1)+2} \dots v_n$ . For example,  $u = \mathbf{a b a}$  can be embedded into  $v = \mathbf{a b a c b b a}$  in six different ways, but the canonical embedding maps  $u$  to the prefix  $v[1..3]$ . This makes it often rather simple to deal with subsequences algorithmically: matching can be decided greedily in linear time; the set of all subsequences of a string  $v$  can be represented by a *deterministic* automaton of size  $O(|v||\Sigma|)$  (which means that the analysis problems can be solved in polynomial time, although much more efficient methods exist in certain cases [33]).

For practical scenarios, on the other hand, it seems reasonable to also postulate some properties with respect to the *gaps* that are induced by the embedding. For example, if we model the scheduling of several threads on a single processor by shuffling several sequences into one string, then a-priori knowledge about the scheduling strategy may tell us that the subsequences describing the single threads will not have huge gaps (any kind of fairness property of the scheduling strategy implies this). Another example is finding alignments of bio-sequences by computing longest common subsequences. While any common subsequence of two strings can be interpreted as an alignment, it is questionable if this interpretation is still useful if roughly half of the positions of the common subsequence are mapped to the beginning of the strings, while the other half is mapped to the end of the strings, with a huge gap (say thousands of symbols) in between. This situation should rather be seen as two individual alignments. In fact, in this scenario the optimisation goal of finding a *longest* common subsequence, without further constraints, even seems counterproductive, since it may favour alignments that are to a large extent disconnected and are therefore less likely

to describe relevant properties. In the context of complex event processing, it might be desirable to describe the situation that between the events of a job  $A$  only events associated to a job  $B$  appear (e. g., due to unknown side-effects this leads to a failure of job  $A$ ). In this case, we are interested in embedding a string as a subsequence such that the gaps only contain symbols from a certain subset of the alphabet (i. e., the events associated to job  $B$ ). So, in practice, it makes sense to reason both about the length and the actual content of gaps induced by embeddings.

The large algorithmic tool box for problems based on subsequences is not always capable of handling the practically relevant scenarios, where we are interested in subsequences that can be embedded not just in *any* way, but in *some specific way* that is reasonable for the application scenario. We therefore investigate basic problems on subsequences in the setting where the gaps of the subsequences (or rather of the embeddings) have certain constraints.

**Related Work.** Subsequences with various types of gap constraints are considered in different contexts. Not unexpectedly, one of the main areas in which such subsequences were investigated is combinatorial pattern matching with biological motivations, see [10] and the references therein. In [49, 50], mining such subsequences is presented as a typical data-mining problem with applications in classification and clustering algorithms. In [44], a query class for event streams is introduced, which is based on subsequences with upper and lower length bounds as gap constraints. The longest common subsequence problem has also been extended to the case where the gaps have length constraints (see, e. g., [38] and the references therein).

Coming back to [10], a rather well-researched problem that is related to our setting is that of matching *variable length gap patterns*. In this setting, a pattern, defined as the string  $u_1(p_1, q_1)u_2(p_2, q_2) \dots u_{m-1}(p_{m-1}, q_{m-1})u_m$  with  $u_i \in \Sigma^+$ ,  $(p_i, q_i) \in \mathbb{N}^2$  and  $0 \leq p_i \leq q_i$ , matches a string  $w$  if  $w = w_0u_1w_1 \dots u_{m-1}w_{m-1}u_mw_m$  with  $p_i \leq |w_i| \leq q_i$ . For this special pattern matching problem many algorithmic results exist (see [10] and the references therein); moreover, it has also been investigated in more practical papers that provide experimental evaluations of algorithms solving it, see, e. g., [7, 17]. The above can be seen as special variants of the matching problem for subsequences with gap-length constraints.

While the works above address mostly patterns with length constraints, the area of string constraint solving (with applications in formal verification, and a strong algorithm engineering component, see [3]) addresses the problem of aligning two strings containing constants (or contiguous sequences of one or more letters) and variables (or gaps). In general (see the aforementioned survey [3] and the references therein), the variables/gaps are subject to conjunctions of pairwise string-equality, length, or regular constraints (see also the discussion in this paper's full version [20]). Moreover, the problem of checking whether factors of words are part of a given regular language were addressed in the context of sliding window algorithms [27, 28, 29, 30, 31] or in the streaming model [9, 22].

On the other hand, we are not aware of any works that are concerned with (non-trivial) gap-constrained variants of the analysis problems (i. e., universality, containment, and equivalence). Let us now formally define the setting considered in this paper.

**Subsequences With Gap Constraints.** Since the gaps induced by an embedding are essentially strings (or words), it seems natural to formalise gap constraints for length- $k$  subsequences by  $(k - 1)$ -tuples of sets of strings (i. e., languages)  $gc = (C_1, \dots, C_{k-1})$ , where  $C_i \subseteq \Sigma^*$  for every  $i \in \{1, \dots, k - 1\}$ ; we denote  $|gc| = k - 1$ . A length- $k$  subsequence  $u = u_1u_2 \dots u_k$  of  $v = v_1v_2 \dots v_n$  satisfies  $gc$  (i. e., it is a  $gc$ -subsequence) if  $u \preceq_e v$  for an embedding  $e$  that satisfies  $gc$  in the sense that, for every  $i \in \{1, \dots, k - 1\}$ ,  $v_{e(i)+1} \dots v_{e(i+1)-1} \in C_i$ . By  $SubSeq(gc, v)$  we denote the set of all  $gc$ -subsequences of  $v$ . In this setting, we consider:

- the *matching problem* MATCH: decide, for given strings  $w, w'$ , and gap constraints  $gc$  with  $|gc| = |w| - 1$ , whether  $w$  is a  $gc$ -subsequence of  $w'$  (i. e., whether  $w \in SubSeq(gc, w')$ );
- the *universality problem* UNI: decide, for given string  $w$  and gap constraints  $gc$  with  $|gc| = k - 1$ , whether  $SubSeq(gc, w) = \Sigma^k$ ;
- the *equivalence problem* EQU (respectively, the *containment problem* CON): decide, for given strings  $w, w'$ , and gap constraints  $gc$ , whether  $SubSeq(gc, w) = SubSeq(gc, w')$  (respectively,  $SubSeq(gc, w) \subseteq SubSeq(gc, w')$ ).

Our formalisation of gap constraints is as general as possible. In order to obtain meaningful results we focus on *regular constraints*, where each  $C_i$  is a regular language, represented by a finite automaton accepting it, and on *length constraints*, where each  $C_i$  has the form  $\{v \in \Sigma^* \mid L^-(i) \leq |v| \leq L^+(i)\}$  with  $L^-(i), L^+(i) \in \mathbb{N} \cup \{0, +\infty\}$  and is represented as the pair  $(L^-(i), L^+(i))$ . We also consider *conjunctions*  $((L^-(i), L^+(i)), R_i)$  of *length and regular constraints*, i. e., the gap must be of length between  $L^-(i)$  and  $L^+(i)$  and from the regular language  $R_i$  (note that simply “pushing” the length constraint into the regular language  $R_i$  would increase the size of  $R_i$ ’s representation as automaton by a factor  $L^+(i)$ , which is exponential in  $L^+(i)$ ’s binary representation). These constraints cover the existing cases in the literature.

**Our Contribution.** We provide a comprehensive picture of the computational complexity of both the matching and the analysis problems, proving tight upper and lower bounds for them, with a focus on the latter.

With respect to matching, we show that we can check whether  $u$  is a  $gc$ -subsequence of  $v$  in *rectangular* time  $O(|v||gc|)$ , where, if each  $C_i$  is the conjunction of a regular constraint and a length constraint,  $|gc|$  is the number of states of the DFAs that represent the regular constraints. In the absence of regular constraints (so, for length constraints only), such rectangular upper bounds are already reported in the literature (see [38]). Moreover, the case when length constraints are absent (so, we have regular constraints only) is rather straightforward. Our algorithm dealing with the case of conjunctions of regular and length constraints requires, however, a non-trivial extension of the existing approaches. Nevertheless, our main contribution in this area is that we can also prove a conditional lower bound that essentially states that these running times of those algorithms cannot be improved unless the *orthogonal vectors hypothesis* fails. More precisely, adding length or regular constraints to subsequences changes the matching problem from a trivial problem to a problem with provably rectangular complexity. Additionally, this proves also a conditional lower bound for matching variable length gap patterns (mentioned above), for which many upper bounds, but no matching lower bound were known before. It is also worth noting that the lower bound holds for the case of a constant alphabet and constant length constraints.

With respect to the problems of universality, equivalence, and containment, we show strong intractability results for both the cases of length constraints and of regular constraints. More precisely, these problems are NP-complete even for a fixed binary alphabet and for small, constant length (or regular) constraints (note that the problems are trivial for a unary alphabet). Moreover, for any fixed constant alphabet, the problems can be solved by brute-force algorithms in exponential time  $2^{O(k)}|gc|\ell$  (recall that  $k$  is the length of subsequences;  $\ell$  is the maximum length of the input strings), and we can show that for alphabets of size at least 3, the exponent can neither be lowered to any  $o(k)$  (unless the *exponential time hypothesis* fails), nor to  $k(1 - \epsilon)$  for any  $\epsilon \geq 1$  (unless the *strong exponential time hypothesis* fails), and these lower bounds even hold for small constant length constraints. If we parameterise by both  $|\Sigma|$  and  $k$ , then the brute-force algorithm is a trivial fpt-algorithm

(i.e., an algorithm whose running time is  $f(k, |\Sigma|)n^{O(1)}$ , for some computable function  $f$ , so an algorithm which is *fixed parameter tractable*, or *fpt-algorithm* for short). However, we can exclude fpt-running times for the cases where we parameterise by only  $|\Sigma|$ , or by only  $k$  (based on the assumptions  $P \neq NP$  and  $FPT \neq W[1]$ , respectively). Note that for classical subsequences all these problems can be easily solved in polynomial time, so our results emphasise the fundamentally different nature of constrained subsequences.

As a last remark, for space reasons, missing proofs and additional comments are only presented in the full version of this paper [20].

## 2 Preliminaries

Let  $\mathbb{N} = \{1, 2, \dots\}$  and  $[n] = \{1, \dots, n\}$  for  $n \in \mathbb{N}$ . By  $\mathcal{P}(S)$ , we denote the power set of a set  $S$ .

For a finite alphabet  $\Sigma$ ,  $\Sigma^+$  denotes the set of non-empty words over  $\Sigma$  and  $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$  (where  $\varepsilon$  is the empty word). For a word  $w \in \Sigma^*$ ,  $|w|$  denotes its length (in particular,  $|\varepsilon| = 0$ ); for every  $b \in \Sigma$ ,  $|w|_b$  denotes the number of occurrences of  $b$  in  $w$ ; we set  $w^1 = w$  and  $w^k = ww^{k-1}$  for every  $k \geq 2$ . For a string  $w = w_1w_2 \dots w_n$  with  $w_i \in \Sigma$  for every  $i \in [n]$ , and for every  $i, j \in [|w|]$  with  $i \leq j$ , we define  $w[i..j] = w_iw_{i+1} \dots w_j$ ; moreover, we use  $w[i]$  as shorthand for  $w[i..i]$ . For any string  $w \in \Sigma^*$ , we define  $\text{alph}(w) = \{b \in \Sigma \mid |w|_b \geq 1\}$ . A *factor* of a string  $w \in \Sigma^*$  is a string  $v \in \Sigma^*$  such that  $w = uvu'$  for  $u, u' \in \Sigma^*$ ; if  $u = \varepsilon$ , then  $v$  is called a *prefix* of  $w$ , and if  $u' = \varepsilon$ , then  $v$  is called a *suffix* of  $w$ .

By REG, we denote the class of regular languages (see [37] for more details). For the considered algorithmic problems we use as computational model the standard unit-cost RAM with logarithmic word size, with inputs over integer alphabets (see this paper's full version [20]).

**Hypotheses.** We now recall some basic computational problems and respective algorithmic hypotheses. We shall use these hypotheses to obtain our conditional lower bounds.

The problem CNF-SAT gets as input a Boolean formula  $F$  in conjunctive normal form as a set of clauses  $F = \{c_1, c_2, \dots, c_m\}$  over a set of variables  $V = \{v_1, v_2, \dots, v_n\}$ , i. e., for every  $i \in [m]$ , we have  $c_i \subseteq \{v_1, \neg v_1, \dots, v_n, \neg v_n\}$ . The question is whether  $F$  is satisfiable. By  $k$ -CNF-SAT, we denote the variant where  $|c_i| \leq k$  for every  $i \in [m]$ .

The *Orthogonal Vectors problem* (OV for short) is defined as follows: Given sets  $A, B$  each containing  $n$  Boolean-vectors of dimension  $d$ , check whether there are vectors  $\vec{a} \in A$  and  $\vec{b} \in B$  that are orthogonal, i. e.,  $\vec{a}[i] \cdot \vec{b}[i] = 0$  for every  $i \in [d]$ .

We shall use the following algorithmic hypotheses based on CNF-SAT and OV that are common for obtaining conditional lower bounds in fine-grained complexity (see the literature mentioned below for further details). In the following, poly is any fixed polynomial function.

- *Exponential Time Hypothesis* (ETH) [40, 51]: 3-CNF-SAT cannot be solved in time  $2^{o(n)} \text{poly}(n+m)$ .
- *Strong Exponential Time Hypothesis* (SETH) [39, 65]: For every  $\epsilon > 0$  there exists a  $k$  such that  $k$ -CNF-SAT cannot be decided in  $O(2^{n(1-\epsilon)}) \text{poly}(n)$ .
- *Orthogonal Vectors Hypothesis* (OVH) [12, 13, 65]: For every  $\epsilon > 0$  there is no algorithm solving OV in time  $O(n^{2-\epsilon} \text{poly}(d))$ .

**Subsequences With Gap Constraints.** We now define subsequences with gap constraints (see also the introduction). In the following, let  $\Sigma$  be a finite *alphabet*. Recall that for a string  $w$ , an embedding is a function  $e : [k] \rightarrow [|w|]$  such that  $i < j$  implies  $e(i) < e(j)$  for all

$i, j \in [k]$ , and it induces the subsequence  $\text{subseq}_e(w) = w[e(1)]w[e(2)] \dots w[e(k)]$  of  $w$ . For every  $j \in [k-1]$ , the  $j^{\text{th}}$  gap of  $w$  induced by  $e$  is the string  $\text{gap}_e(w, j) = w[e(j)+1..e(j+1)-1]$ . We say that  $e$  is the embedding of  $\text{subseq}_e(w)$  in  $w$ .

An  $\ell$ -tuple of gap constraints is a tuple  $gc = (C_1, C_2, \dots, C_\ell)$  with  $C_i \subseteq \Sigma^*$  for every  $i \in [\ell]$ . For convenience, we set  $gc[i] = C_i$  for every  $i \in [\ell]$ . We say that an embedding  $e$  satisfies a  $(k-1)$ -tuple of gap constraints  $gc$  with respect to a string  $w$  if it has the form  $e : [k] \rightarrow [|w|]$ , and, for every  $i \in [k-1]$ ,  $\text{gap}_e(w, i) \in C_i$ . Moreover, for a  $(k-1)$ -tuple  $gc$  of gap constraints, the set  $\text{SubSeq}(gc, w)$  contains all subsequences of  $w$  induced by embeddings that satisfy  $gc$ , i. e.,  $\text{SubSeq}(gc, w) = \{\text{subseq}_e(w) \mid e \text{ is an embedding that satisfies } gc \text{ w. r. t. } w\}$ . The elements of  $\text{SubSeq}(gc, w)$  are also called the  $gc$ -subsequences of  $w$ . Note that tuples of gap constraints do not have constraints for the prefix  $w[1..e(1)]$  or suffix  $w[e(k)..|w|]$ . However, our formalism can model this case too (for details, see this paper's full version [20]). For a  $(|u|-1)$ -tuple  $gc$  of gap constraints, we write  $u \preceq_{gc} v$  to denote that  $u \preceq_e v$  for some embedding  $e : [|u|] \rightarrow [|v|]$  that satisfies  $gc$  with respect to  $v$ , i. e.,  $u \preceq_{gc} v$  means that  $u$  is a  $gc$ -subsequence of  $v$ . We note that for tuples of gap constraints  $gc = (C_1, C_2, \dots, C_{k-1})$  with  $C_i = \Sigma^*$  for every  $i \in [k-1]$ , the set  $\text{SubSeq}(gc, w)$  is just the set of all length- $k$  subsequences of  $w$ .

**Special Types of Gap Constraints.** We now define the types of gap constraints that are relevant for our work. We say that the gap constraints  $gc = (C_1, \dots, C_{k-1})$  are

- *regular constraints* if  $C_i \in \text{REG}$  for every  $i \in [k-1]$ . For every  $i \in [k-1]$ , we represent the regular constraint  $C_i$  by a deterministic finite automaton (for short, DFA)  $A_i$  accepting it. See this paper's full version [20] for a discussion on the choice of DFAs to represent regular constraints.
- *length constraints* if, for every  $i \in [k-1]$ , there are  $L^-(i), L^+(i) \in \mathbb{N} \cup \{0, +\infty\}$  with  $L^-(i) \leq L^+(i)$ , such that  $C_i = \{v \in \Sigma^* \mid L^-(i) \leq |v| \leq L^+(i)\}$ . We represent length constraints succinctly by pairs of numbers  $(L^-(i), L^+(i))$ ,  $i \in [k-1]$ , in binary encoding.
- *reg-len constraints* if, for every  $i \in [k-1]$ ,  $C_i$  is the conjunction of a regular constraint  $C'_i$  and a length constraint  $(L^-(i), L^+(i))$ , i. e.,  $C_i = C'_i \cap \{v \in \Sigma^* \mid L^-(i) \leq |v| \leq L^+(i)\}$ . We represent such constraints by  $((L^-(i), L^+(i)), A'_i)$ , where  $A'_i$  is a DFA accepting  $C'_i$ .

A gap constraint  $C_i$  is a *zero-gap* if and only if  $C_i = \{\varepsilon\}$ . Let  $\text{nz}(gc)$  be the number of non-zero-gaps of  $gc$  (that is, the number of positions  $i$  such that  $C_i \neq \{\varepsilon\}$ ). For a tuple of regular or reg-len gap constraints  $gc$ , let  $\text{size}(gc)$  be the size of the overall representation of the respective constraints (total size of the automata defining the constraints) and let  $\text{states}(gc)$  be the total number of states of the DFAs  $A_i$ , for  $i \in [k-1]$ , corresponding to the non-zero gaps of  $gc$ .

Clearly, length constraints are the simplest type of gap constraints considered above. In particular, length constraints, and therefore reg-len constraints, can also be seen as a particular case of regular constraints. However, transforming length or reg-len constraints into a single automaton may cause an exponential size increase.

**Problems for Subsequences With Gap Constraints.** In this paper, we investigate the matching problem `MATCH` and the analysis problems `UNI`, `CON`, and `EQU` (see definitions in the introduction). For simplicity, the pairs  $(p, gc)$ , which play the role of the patterns in `MATCH`, will be called *gap-constrained sequences*, or simply *gapped sequences* for short. By `MATCH $_\Sigma$` , we denote the problem variant where all instances are over the fixed alphabet  $\Sigma$ ; for some class  $\mathcal{C}$  of gap constraints, we use “`MATCH` with  $\mathcal{C}$ -constraints” to refer to the variant where the constraints are from  $\mathcal{C}$ . We use analogous notations for the analysis problems.

If  $gc = (\Sigma^*, \Sigma^*, \dots, \Sigma^*)$ , then MATCH boils down to the simple task of checking whether a given string is a subsequence of another string. The equivalence problem for such trivial gap constraints, on the other hand, boils down to the well-known problem of deciding the Simon congruence for two strings (see the discussion in the introduction). Our setting naturally models many other classical problems; some are discussed in Section 5. Finally, even though our framework allows arbitrary gap constraints, we will stick to the specific natural and relevant types of constraints defined above (i. e., length, regular, reg-len constraints).

### 3 Matching Gapped Subsequences

This section contains two main results. Firstly, we show that MATCH with reg-len constraints can be solved in  $O(|w| \text{states}(gc) + \text{size}(gc))$  time, which implies also rectangular upper bounds for MATCH with either length or regular constraints. Secondly, we show that, assuming OVH holds, there are no algorithms solving any of these problems polynomially faster.

Note that, when dealing with length constraints, a constraint  $(L^-(i), L^+(i))$  is equivalent to the regular language  $C_i = \{x \in \Sigma^* \mid L^-(i) \leq |x| \leq L^+(i)\}$ , which is accepted by a DFA with  $\Theta(L^+(i))$  states. So, we could also interpret a tuple  $gc$  of reg-len constraints as a tuple of regular constraints only, by considering in each component of  $gc$  the intersection of the regular constraint with the regular language defined by the length constraints. However, this would lead to a growth in the number of states needed to model  $gc$ , and, as we will see in the following, to a less efficient algorithm for MATCH. In this setting, we state our first main result. The full proof is given in the full version of this paper [20]. To emphasise the merits of our approach, we overview in the full version of this paper [20] also several simpler approaches and their complexity (and shortcomings).

► **Theorem 1.** *MATCH with reg-len constraints can be solved in  $O(|w|(\text{states}(gc) + 1) + \text{size}(gc))$  time.*

**Proof Sketch.** Assume  $|w| = n$ ,  $|p| = m$ , and  $gc = ((L^-(1), L^+(1)), A_1), \dots, (L^-(m-1), L^+(m-1)), A_{m-1})$ , where  $A_i = (Q_i, q_{0,i}, F_i, \delta_i)$  are DFAs for the regular constraints and  $(L^-(i), L^+(i))$  are the length constraints. Let  $i_1, \dots, i_{k-1} \in [m-1]$  be such that  $C_i \neq \{\varepsilon\}$  (i. e.,  $C_i$  is a non-zero constraint of  $gc$ ), for all  $i \in \{i_1, \dots, i_{k-1}\}$ , and  $C_i = \{\varepsilon\}$ , for all  $i \notin \{i_1, \dots, i_{k-1}\}$ . Clearly,  $\text{states}(gc) = \sum_{j=1}^{k-1} |Q_{i_j}| \geq \text{nz}(gc)$ . With  $i_0 = 0$  and  $i_k = m$ , we compute the words  $p_j = p[(i_{j-1} + 1)..i_j]$ , for  $j \in [k]$ , and we construct in linear time longest common extension data structures (see [21]) for the word  $x = wp$ , allowing us to check in constant time whether  $w[i + 1..i + |p_j|] = p_j$ , for  $i \leq n$ .

The main part of our algorithm consists in a dynamic programming approach. We compute a two-dimensional  $n \times k$  array  $D[\cdot][\cdot]$  (initialised with 0), where  $D[i][\ell] = 1$  iff  $p[1..|p_1| \dots p_\ell]$  can be embedded in  $w[1..i]$  such that the first  $\ell - 1$  non-zero constraints are satisfied, and  $p_\ell$  is mapped to  $w[1..i]$ 's length- $|p_\ell|$  suffix. Otherwise,  $D[i][\ell] = 0$ . We first set  $D[i][1] = 1$  iff  $w[i - |p_1| + 1..i] = p_1$ . Further, assume that, for some  $t \in [k - 1]$ , we have computed  $D[\cdot][\ell]$ , for all  $\ell \leq t$ , and we want to compute  $D[\cdot][t + 1]$ . The main component of this most involved part is computing an array  $f_{t+1}[\cdot]$ , with  $n$  elements, such that  $f_{t+1}[i] = 1$  iff there exists a position  $j$  for which  $D[j][t] = 1$ ,  $w[j + 1..i] \in L(A_t)$ , and  $L^-(t) \leq |w[j + 1..i]| \leq L^+(t)$ . We now sketch this step (a full description is given in the full version of this paper [20]).

We first collect in a list  $L_{t+1} = j_1 < \dots < j_r$  (increasingly sorted) all the positions  $j$  of  $w$  with  $D[j][t] = 1$ . We then compute a graph  $G_{t+1}$  with nodes  $(i, q)$ , with  $i \in [n]$  and  $q \in Q_t$ , that consists of the union, over  $j \in L_{t+1}$ , of the (not necessarily disjoint) paths

$[(j, q_{0,t}), (j+1, q_1^j), \dots, (n, q_{n-j}^j)]$ , where  $\delta_t(q_{0,t}, w[j+1]) = q_1^j$  and  $\delta_t(q_r^j, w[j+r+1]) = q_{r+1}^j$ , for all  $r \in [n-j-1]$ . Intuitively, such a path records the trace of the computation of  $A_t$  on the input  $w[j+1..n]$ . These paths (and, therefore, the graph  $G_{t+1}$ ) can be simultaneously constructed to avoid redundant computations.  $A_t$  is deterministic, thus, if two such paths intersect, then they are identical after their first common node. Consequently,  $G_{t+1}$  is a collection of disjoint trees  $T_1, T_2, \dots, T_z$ . As there are no edges between any pair of nodes  $(n, q)$  and  $(n, q')$ , with  $q, q' \in Q_t$ , each such tree  $T_i$  can be seen as a rooted tree, whose root is its single node of the form  $(n, q)$  and whose leaves are some of the nodes  $(j, q_{0,t})$ , with  $j \in L_{t+1}$ .

For each tree  $T_i$  and for each leaf  $(j, q_{0,t})$  of  $T_i$ , we mark all the ancestors  $(d, q)$  of  $(j, q_{0,t})$  such that  $L^-(t) \leq |w[j+1..d]| = d-j \leq L^+(t)$ . After this, a node  $(j, q)$  of  $T_i$  is marked iff there exists a length- $\ell$  path  $\mathcal{P}$ , with  $L^-(t) \leq \ell \leq L^+(t)$ , which connects a leaf  $(j', q_{0,t})$  of  $T_i$  to  $(j, q)$ , i. e.,  $\delta_t(q_{0,t}, w[j'+1..j]) = q$ . By using efficient data structures, computing and marking the trees takes time  $O(n|Q_t|)$  and  $O(\sum_{i=1}^p |T_i|)$ , resp.

Finally, for  $i = 1, \dots, n$ , we set  $f_{t+1}[i] = 1$  iff there is a state  $q \in F_t$  such that node  $(i, q)$  is marked. This means that  $f_{t+1}[i] = 1$  iff there is a word  $w[j+1..i]$  of length  $\ell$ , with  $L^-(t) \leq \ell \leq L^+(t)$ , such that  $j \in L_{t+1}$  and  $\delta_t(q_{0,t}, w[j+1..i])$  is a final state (i. e.,  $w[j+1..i] \in L(A_t)$ ).

For computing the elements of  $D$ , we set  $D[i][t+1] = 1$  iff  $w[i - |p_{t+1}| + 1..i] = p_{t+1}$  and  $f_{t+1}[i - |p_{t+1}|] = 1$ . Then, we decide that  $p \preceq_{gc} w$  iff there exists  $j$  with  $D[j][k] = 1$ .

The whole process can be implemented in  $O(|w| \text{states}(gc) + \text{size}(gc))$  time. ◀

The next results are now immediate. Note that for these particular cases (but, to the best of our knowledge, not for their conjunction, covered in Theorem 1) simpler algorithms exist.

► **Corollary 2.**

- (1) *MATCH with length constraints can be solved in  $O(|w|(\text{nz}(gc) + 1))$  time.*
- (2) *MATCH with regular constraints can be solved in  $O(|w|(\text{states}(gc) + 1) + \text{size}(gc))$  time.*

It is worth noting that the matching problem can be solved in  $O(|w|)$  time when  $gc$  only defines constraints that are  $\{\varepsilon\}$  or  $\Sigma^*$ , which covers, e. g., the cases of subsequence matching or string matching. In particular, the greedy strategy used for matching regular patterns with variables (see, e. g., [23]) can be easily adapted to solve MATCH with length constraints in linear time, when the upper bounds on each gap are trivial (i. e., they are all greater or equal to the length of the input word). So, as far as length constraints are concerned, it seems that non-trivial upper bounds lead to an increase in the difficulty of the MATCH problem; a particularly efficient approach for subsequences with general length constraints is given in [10], but, in the worst case, it still has rectangular complexity. However, even when non-trivial length upper bounds are used, there are still some simpler particular cases. For instance, when working with *strings with don't cares* (or partial words), where each gap has a fixed length (i. e., the lower and upper bounds are the same), MATCH can be solved in time  $O(|w| \log |p|)$  [18].

A gapped sequence  $(p, gc)$  with reg-len constraints can be represented as a classical regular expression  $r_{(p,gc)}$ , so MATCH can be solved by a textbook algorithm in  $O(|w||r_{(p,gc)}|)$  [62], which is optimal w. r. t. polynomial speed-ups, conditional on OVH [6]. However, including the string  $p$  and the length constraints in the regular expression might, once more, lead to a slower algorithm compared to our direct approach, as  $|r_{p,gc}|$  may be much larger than  $\text{states}(gc)$ .



To summarise, at an intuitive level, we could say that as long as we have non-trivial length or regular constraints, MATCH seems to become more difficult than its counterpart for classical subsequences. This intuitive remark is confirmed by our second main result.

► **Theorem 3.** *MATCH with length constraints cannot be solved in  $\mathcal{O}(|w|^h \text{nz}(gc)^g)$  time with  $h + g = 2 - \epsilon$  for some  $\epsilon > 0$ , unless OVH fails. This holds even if  $|\Sigma| = 4$  and all length constraints are  $(0, \ell)$  with  $\ell \leq 6$ .*

**Proof Sketch.** Let  $A = \{\vec{a}_1, \dots, \vec{a}_n\}$  and  $B = \{\vec{b}_1, \dots, \vec{b}_n\}$ , with  $A, B \subset \{0, 1\}^d$  be an OV-instance. We transform  $A$  into a string  $w \in \Sigma^* = \{0, 1, \#, @\}^*$  and  $B$  into a string  $p \in \Sigma^*$  and a  $(|p| - 1)$ -tuple  $gc$  of length constraints. For convenience, we represent the gapped sequence  $(p, gc)$  with  $p = p[1] \cdots p[m]$  by writing the length constraints in between the symbols, i. e.,  $p[1] \xleftrightarrow{gc[1]} p[2] \xleftrightarrow{gc[2]} \cdots \xleftrightarrow{gc[m-1]} p[m]$ , and we omit  $\xleftrightarrow{gc[i]}$  if  $gc[i] = (0, 0)$ . For example, if  $p = abab$  and  $gc[1] = (0, 0)$ ,  $gc[2] = (1, 5)$ , and  $gc[3] = (0, 6)$ , we use the notation  $ab \xleftrightarrow{(1,5)} a \xleftrightarrow{(0,6)} b$ .

Let  $\vec{a}_i = (a_i^1, \dots, a_i^d)$  and  $\vec{b}_i = (b_i^1, \dots, b_i^d)$ , for all  $i \in [n]$ . We shall represent the vectors from  $A$  and  $B$  by different encodings  $C_a(\cdot)$  and  $C_b(\cdot)$ , respectively. The 0 and 1 entries in the  $A$ -vectors are encoded by  $C_a(0) = 010$  and  $C_a(1) = 100$ , and the 0 and 1 entries in the  $B$ -vectors are encoded by  $C_b(0) = 10$  and  $C_b(1) = 01$ . We note that for every  $x, y \in \{0, 1\}$ ,  $C_b(x)$  is a factor of  $C_a(y)$  if and only if  $x \cdot y = 0$ . This means that the orthogonality of  $\vec{a}_i$  and  $\vec{b}_i$  is characterised by the situation that, for every  $j \in [d]$ ,  $C_b(b_i^j)$  is a factor of  $C_a(a_i^j)$ .

We represent each bit  $a_i^j$  of  $\vec{a}_i \in A$  as the string  $\# C_a(0) \# \# C_a(a_i^j) \# \# C_a(0) \#$ , and the vector  $\vec{a}_i$  as the concatenation  $C_a(\vec{a}_i) = \prod_{j=1}^d ([1 \# C_a(0) \#]_1 [2 \# C_a(a_i^j) \#]_2 [3 \# C_a(0) \#]_3)$ , where the brackets  $[1 \dots]_1, [2 \dots]_2, [3 \dots]_3$  are not actual symbols of the gadget, but serve the only purpose to illustrate that  $C_a(\vec{a}_i)$  has three individual *tracks*, where track 1 and 3 correspond to  $d$  occurrences of  $\# C_a(0) \#$  (representing the all-0 vector), while track 2 represents the actual vector  $\vec{a}_i$ . These three tracks play a central role in the correctness of the reduction.

For  $i \in [n]$ , every vector  $\vec{b}_i \in B$  is also represented by listing all bit encodings  $C_b(b_i^j)$ , but in a slightly different way and, most importantly, as a gapped sequence (in the notation defined above):  $(C_b(\vec{b}_i), gc_i) = \left( \prod_{j=1}^{d-1} (\# \xleftrightarrow{\leq 1} C_b(b_i^j) \xleftrightarrow{\leq 1} \# \# \xleftrightarrow{\leq 3} \# \# \xleftrightarrow{\leq 3} \#) \right) \# \xleftrightarrow{\leq 1} C_b(b_i^d) \xleftrightarrow{\leq 1} \#$ .

It can be shown (see the full version of this paper [20]) that if  $C_b(\vec{b}_i) \preceq_e C_a(\vec{a}_\ell)$  and  $e$  satisfies  $gc_i$ , then the embedding  $e$  maps each  $C_b(b_i^j)$  to the  $C_a(0)$  of  $C_a(\vec{a}_\ell)$ 's first track, or each  $C_b(b_i^j)$  to the  $C_a(a_\ell^j)$  of  $C_a(\vec{a}_\ell)$ 's second track, or each  $C_b(b_i^j)$  to the  $C_a(0)$  of  $C_a(\vec{a}_\ell)$ 's third track. More precisely, due to how we use the symbols  $\#$ , the factor  $C_b(b_i^1)$  must be mapped to  $[1 \# C_a(0) \#]_1$  or to  $[2 \# C_a(a_\ell^1) \#]_2$  or to  $[3 \# C_a(0) \#]_3$ . Since we have 4 occurrences of  $\#$  between each  $C_b(b_i^j)$  and  $C_b(b_i^{j+1})$ , and between two consecutive parts of the same track in  $C_a(\vec{a}_i)$ , all the following factors  $C_b(b_i^2), C_b(b_i^3), \dots$  must be mapped to the same track  $C_b(b_i^1)$  is mapped to. This is illustrated in Figure 1. Based on these considerations, it is clear that  $C_b(\vec{b}_i) \preceq_e C_a(\vec{a}_\ell)$  with  $e$  mapping  $C_b(\vec{b}_i)$  to  $C_a(\vec{a}_\ell)$ 's second track is possible if and only if  $\vec{a}_\ell$  and  $\vec{b}_i$  are orthogonal.

$C_b(\vec{b}_i) =$	$\#$	$\xleftrightarrow{\leq 1} C_b(b_i^1) \xleftrightarrow{\leq 1}$	$\#$	$\#$	$\xleftrightarrow{\leq 3}$	$\#$	$\#$	$\xleftrightarrow{\leq 3}$	$\#$	$\#$	$\xleftrightarrow{\leq 1} C_b(b_i^2) \xleftrightarrow{\leq 1}$	$\#$	$\#$	$\xleftrightarrow{\leq 3}$	$\#$	$\dots$
$C_b(\vec{b}_i) =$			$\#$	$\xleftrightarrow{\leq 1} C_b(b_i^1) \xleftrightarrow{\leq 1}$	$\#$	$\#$	$\xleftrightarrow{\leq 3}$	$\#$	$\#$	$\xleftrightarrow{\leq 3}$	$\#$	$\#$	$\xleftrightarrow{\leq 1} C_b(b_i^2) \xleftrightarrow{\leq 1}$	$\#$	$\dots$	
$C_b(\vec{b}_i) =$					$\#$	$\xleftrightarrow{\leq 3} C_b(b_i^1) \xleftrightarrow{\leq 3}$	$\#$	$\#$	$\xleftrightarrow{\leq 3}$	$\#$	$\#$	$\xleftrightarrow{\leq 3}$	$\#$	$\dots$		
$C_a(\vec{a}_\ell) =$	$\#$	$C_a(0)$	$\#$	$\#$	$C_a(a_\ell^1)$	$\#$	$\#$	$C_a(0)$	$\#$	$\#$	$C_a(0)$	$\#$	$\#$	$C_a(a_\ell^1)$	$\#$	$\dots$

■ **Figure 1** Possible embeddings of  $C_b(\vec{b}_i)$  in  $C_a(\vec{a}_\ell)$ , selecting its first, second, or third track.

The remaining challenge is to combine the gadgets  $C_a(\vec{a}_i)$  into a string  $w$ , and the gadgets  $(C_b(\vec{b}_i), gc_i)$  into a gapped sequence  $(p, gc)$ , such that  $p \preceq_e w$  for an embedding  $e$  satisfying  $gc$  if and only if  $e$  is such that every  $(C_b(\vec{b}_i), gc_i)$  is mapped to some  $C_a(\vec{a}_\ell)$ , and there is

## 64:10 Subsequences with Gap Constraints

necessarily at least one pair  $i, \ell \in [n]$  such that  $(C_b(\vec{b}_i), gc_i)$  is embedded into  $C_a(\vec{a}_\ell)$ 's second track. We next define  $w$  and  $(p, gc)$ , and then discuss why they satisfy the property from above:

$$w = \left( \prod_{i=1}^{n-1} @ C_a(\vec{a}_i) \right) @ C_a(\vec{a}_n) \left( \prod_{i=1}^{n-1} @ C_a(\vec{a}_i) \right) @,$$

$$(p, gc) = @ \overset{\leq 5}{\leftrightarrow} \left( \prod_{j=1}^{n-1} C_b(\vec{b}_j) \overset{\leq 1}{\leftrightarrow} \# \overset{\leq 3}{\leftrightarrow} \# \overset{\leq 1}{\leftrightarrow} \# \overset{\leq 3}{\leftrightarrow} \# \overset{\leq 6}{\leftrightarrow} \right) C_b(\vec{b}_n) \overset{\leq 5}{\leftrightarrow} @.$$

If  $p \preceq_e w$  for an embedding  $e$  satisfying  $gc$ , then the first  $@$ -symbol of  $p$  is mapped to the  $@$ -symbol of  $w$  occurring before an occurrence of  $C_a(\vec{a}_{\ell_1})$  for some  $\ell_1$ , and this occurrence is in the prefix  $\left( \prod_{i=1}^{n-1} @ C_a(\vec{a}_i) \right)$  of  $w$ . By reasoning about the occurrences of symbols  $\#$  and the length constraints (see the full version of this paper [20]), we can show that  $C_b(\vec{b}_1)$  must be embedded in  $C_a(\vec{a}_{\ell_1})$  in the way discussed above (i. e.,  $gc_1$  is satisfied and  $C_b(\vec{b}_1)$  is entirely mapped to some track  $q \in \{1, 2, 3\}$  of  $C_a(\vec{a}_{\ell_1})$ ). For simplicity, assume that  $\ell_1 \leq n - 1$ . The factor  $\overset{\leq 1}{\leftrightarrow} \# \overset{\leq 3}{\leftrightarrow} \# \overset{\leq 1}{\leftrightarrow} \# \overset{\leq 3}{\leftrightarrow} \# \overset{\leq 6}{\leftrightarrow}$  between  $(C_b(\vec{b}_1), gc_1)$  and the next part  $(C_b(\vec{b}_2), gc_2)$  will enforce that  $C_b(\vec{b}_2)$  is embedded in  $C_a(\vec{a}_{\ell_1+1})$ , and, moreover, it will be mapped to  $C_a(\vec{a}_{\ell_1+1})$ 's track  $q$  or  $q + 1$  (as there can be at most 18 symbols between  $C_b(\vec{b}_1)$  and  $C_b(\vec{b}_2)$ , track 3 cannot be reached in the case  $q = 1$ ).

By repeating this argument, we can show that if  $(C_b(\vec{b}_j), gc_j)$  is embedded in track  $s$  of  $C_a(\vec{a}_{\ell_j})$  (with  $\ell_j \leq n - 1$ ), then  $(C_b(\vec{b}_{j+1}), gc_{j+1})$  is embedded in track  $s$  or  $s + 1$  of  $C_a(\vec{a}_{\ell_{j+1}})$  in case that  $s \in \{1, 2\}$ , and it is necessarily embedded in track  $s$  of  $C_a(\vec{a}_{\ell_{j+1}})$  in case that  $s = 3$ . If  $\ell_j = n$ , then analogously  $(C_b(\vec{b}_{j+1}), gc_{j+1})$  is mapped to  $C_a(\vec{a}_1)$  of  $w$ 's suffix  $\left( \prod_{i=1}^{n-1} @ C_a(\vec{a}_i) \right) @$ . Consequently, each  $C_b(\vec{b}_j)$  is mapped to a track of  $C_a(\vec{a}_{\ell_j})$ , and the tracks to which these  $C_b(\vec{b}_j)$  are mapped may start with track 1 or 2, and then can only increase until we possibly map some  $C_b(\vec{b}_j)$  to track 3. However, after having mapped the last occurrence of  $\#$  in  $C_b(\vec{b}_n)$  to an occurrence of  $\#$  in  $C_a(\vec{a}_{\ell_n})$ , we can afford a gap of length at most 5 before mapping the last symbol  $@$  of  $(p, gc)$  to an occurrence of  $@$  in  $w$ . By the structure of  $(p, gc)$  and  $w$ , this is only possible if  $C_b(\vec{b}_n)$  is mapped to track 2 or 3 of  $C_a(\vec{a}_{\ell_n})$ .

We conclude that if  $p \preceq_{gc} w$ , then, for some  $j, \ell_j \in [n]$ ,  $(C_b(\vec{b}_j), gc_j)$  is mapped to track 2 of  $C_a(\vec{a}_{\ell_j})$ ; thus,  $\vec{a}_{\ell_j}$  and  $\vec{b}_j$  are orthogonal. On the other hand, the explanations from above show that if  $\vec{a}_{\ell_j}$  and  $\vec{b}_j$  are orthogonal vectors, then  $p$  can be embedded into  $w$  by an embedding that satisfies  $gc$ , i. e., an embedding that maps  $(C_b(\vec{b}_j), gc_j)$  to track 2 of  $C_a(\vec{a}_{\ell_j})$ , all  $(C_b(\vec{b}_{j'}), gc_{j'})$  with  $1 \leq j' < j$  to the first tracks of some  $C_a(\vec{a}_{\ell_{j'}}$ ), and all  $(C_b(\vec{b}_{j'}), gc_{j'})$  with  $j < j' \leq n$  to the third tracks of some  $C_a(\vec{a}_{\ell_{j'}}$ ).

In this reduction, we have  $|\Sigma| = 4$ , all constraints are  $(0, \ell)$  with  $\ell \leq 6$ , and  $|w|, |p| \in \Theta(nd)$ . If MATCH can be solved in  $O(|w|^g |p|^h)$  with  $g + h = 2 - \epsilon$  for some  $\epsilon > 0$ , then OV can be solved in  $O(nd + (nd)^{2-\epsilon})$ . Since  $\text{nz}(gc) \in \Theta(nd)$ , solving MATCH in  $O(|w|^g \text{nz}(gc)^h)$  with  $g + h = 2 - \epsilon$  for some  $\epsilon < 0$  also contradicts OVH.  $\blacktriangleleft$

We emphasise that, according to our proof, these lower bounds hold for MATCH with length constraints even if we only have constant upper bounds on the length of the gaps.

**► Corollary 4.** *MATCH with regular constraints cannot be solved in  $\mathcal{O}(|w|^h \text{states}(gc_p)^g)$  time with  $h + g = 2 - \epsilon$  for some  $\epsilon > 0$ , unless OVH fails. This holds even if  $|\Sigma| = 4$  and all regular constraints are expressed by constant size DFAs.*

From Theorem 3 and Corollary 4, we also get that MATCH with length, regular, or reg-len constraints cannot be solved in  $\mathcal{O}(|w|^h |p|^g)$  time, with  $h + g = 2 - \epsilon$ , nor in  $\mathcal{O}(|w|^{2-\epsilon})$  time. Moreover (see the full version of this paper [20]) we can show similar lower bounds for  $|\Sigma| = 2$  as well.

Compared to the OVH-bound for regular expression matching of [6], we provide a lower bound for a much more restricted problem (i. e., matching gapped sequences with length constraints, a subclass of regular expressions that still seems to have a significant practical relevance); thus, a stronger lower bound (this is also why our OV-reduction has a significantly different structure and is technically more involved than that of [6]). In particular, our lower bound applies (unlike those from [6]) to the case of matching variable length gap patterns, and settles the complexity of that problem. We wrap up this section by noting that Theorem 3 and Corollary 4 show that (if OVH holds) the algorithm of Theorem 1, also when used for regular constraints or length constraints only, is optimal in the sense that there are no algorithms which can solve MATCH in the respective settings polynomially faster.

#### 4 Analysis Problems for Gapped Subsequences

Let us recall that the *universality*, *containment* and *equivalence problem* (denoted by UNI, CON and EQU for short) consist in deciding  $SubSeq(gc, w) = \Sigma^k$ ,  $SubSeq(gc, w) \subseteq SubSeq(gc, w')$ , and  $SubSeq(gc, w) = SubSeq(gc, w')$ , respectively, for a given  $(k - 1)$ -tuple  $gc$  of gap constraints and strings  $w, w' \in \Sigma^*$ . As mentioned before, these problems can be solved in polynomial time for classical subsequences (see the full version of this paper [20] for further details). We show next that these problems become much harder for non-trivial length or regular constraints.

From Corollary 2 and Theorem 1, we can directly conclude the following brute-force upper bounds.

##### ► Theorem 5.

- (1) *The problems UNI, CON and EQU with length (or reg-len) constraints can be solved in time  $O(|\Sigma|^k \text{nz}(gc)\ell)$  (respectively,  $O(|\Sigma|^k \text{states}(gc)\ell)$ ), where  $\ell = \max\{|w|, |w'|\}$ .*
- (2) *The problems  $UNI_\Sigma$ ,  $CON_\Sigma$  and  $EQU_\Sigma$  with length (or reg-len) constraints can be solved in time  $2^{O(k)} \text{nz}(gc)\ell$  (respectively,  $2^{O(k)} \text{states}(gc)\ell$ ), where  $\ell = \max\{|w|, |w'|\}$ .*

We shall next complement these brute-force upper bounds by suitable lower bounds, which demonstrate that significantly faster algorithms are unlikely to exist. For convenience, we state our complexity results for the complement problems, i. e., *non-universality problem* (NUNI), *non-containment problem* (NCON), and *non-equivalence problem* (NEQU). Moreover, we state the lower bounds for the case of length constraints only. By simply interpreting the length constraints as regular constraints, all the lower bounds also apply to the case of regular constraints (this does not cause an exponential size increase of the instances, see the full version of this paper [20]).

Our first result establishes the general NP-completeness (even for small constant alphabets and length constraints), and that the exponent  $O(k)$  of Theorem 5(2) cannot be significantly improved, unless ETH or SETH fail. We will discuss some proof ideas later on.

► **Theorem 6.** *For every fixed alphabet  $\Sigma$  with  $|\Sigma| \geq 3$ ,  $NUNI_\Sigma$ ,  $NCON_\Sigma$  and  $NEQU_\Sigma$  with length constraints are NP-complete, even if all length constraints are (1, 5). Moreover,*

- *they cannot be solved in subexponential time  $2^{o(k)} \text{poly}(|w|, k)$  (unless ETH fails),*
- *they cannot be solved in time  $O(2^{k(1-\epsilon)} \text{poly}(|w|, k))$  (unless SETH fails).*

This directly leads to the question whether these problems are tractable if  $|\Sigma| \leq 2$ . This is obviously true for unary alphabet  $\Sigma = \{\mathbf{a}\}$  (note that in this case,  $SubSeq(gc, w) = \Sigma^k = \{\mathbf{a}^k\}$  if  $(\sum_{i \in [k]} L^-(i)) + k \leq |w|$ , and  $SubSeq(gc, w) = \emptyset$  otherwise), but NP-complete for  $|\Sigma| = 2$ :

$$\left( \begin{array}{ccc} \{a\} & \{b\} & \{a, c, d\} \\ \{c, d\} & \{b\} & \{a\} \\ \{a\} & \{b, c, d, e\} & \{d\} \\ \{e\} & \{b\} & \{c, e\} \end{array} \right) \quad \left( \begin{array}{cccccc} \{0\} & \{1\} & \{0\} & \{0, 1\} & \{0, 1\} & \{0, 1\} \\ \{1\} & \{1\} & \{0, 1\} & \{0, 1\} & \{0, 1\} & \{0\} \\ \{0, 1\} & \{0, 1\} & \{0\} & \{0\} & \{0\} & \{0, 1\} \end{array} \right)$$

■ **Figure 2** Left side: example instance of METANUNI for  $\Gamma = \{a, b, c, d, e\}$ ,  $q = 4$ , and  $k = 3$ . Note that, e. g.,  $W_{3,2} = \{b, c, d, e\}$  and  $W_{4,1} = \{e\}$ ; moreover,  $\mathcal{L}(W_1) = \{a\} \cdot \{b\} \cdot \{a, c, d\} = \{a b a, a b c, a b d\}$ . Since  $\cup_{i \in [4]} \mathcal{L}(W_i) \neq \Gamma^3$ , this is a negative instance. Right side: the CNF-SAT-instance  $c_1 = \{v_1, \neg v_2, v_3\}$ ,  $c_2 = \{\neg v_1, \neg v_2, v_5\}$ ,  $c_3 = \{v_3, v_4, v_5\}$  over the variables  $\{v_1, v_2, \dots, v_6\}$  as an instance of METANUNI. Note that  $100010 \notin \cup_{i \in [3]} \mathcal{L}(W_i)$ ; thus, 100010 is a satisfying assignment.

► **Theorem 7.** *For every fixed alphabet  $\Sigma$  with  $|\Sigma| = 2$ ,  $NUNI_\Sigma$ ,  $NCON_\Sigma$  and  $NEQU_\Sigma$  with length constraints are NP-complete even if each length constraint is  $(0, 0)$  or  $(3, 9)$ .*

Let us now consider the case where  $\Sigma$  is not treated as a constant. Theorem 5 means that NUNI, NCON and NEQU with length constraints are trivially fixed parameter tractable if parameterised by both  $|\Sigma|$  and  $k$ . Moreover, since  $\ell = \max\{|w|, |w'|\}$  bounds both  $|\Sigma|$  and  $k$ , we also have fixed parameter tractability with respect to  $\ell$  for trivial reasons. Are the problems fixed-parameter tractable with respect to the single parameter  $|\Sigma|$  or the single parameter  $k$ ? With respect to  $|\Sigma|$ , this is answered in the negative by Theorem 7 (unless  $P = NP$ ). With respect to parameter  $k$ , the following result gives a negative answer as well.

► **Theorem 8.** *Problems  $NUNI$ ,  $NCON$  and  $NEQU$  with length constraints cannot be solved in running time  $O(f(k) \text{ poly}(|w|, k))$  for any computable function  $f$  (unless  $FPT = W[1]$ ).*

This result only holds for unbounded alphabets and length constraints. Indeed, for constant  $\Sigma$  the brute-force algorithm is an fpt-algorithm with respect to  $k$ . Moreover, if the upper length constraints are bounded by some constant  $\ell$ , then we only have to enumerate at most  $\ell^{k-1}$  candidate tuples of gap sizes and check whether one of them induces an embedding satisfying  $gc$  with respect to  $w$ , which again would yield an fpt-algorithm with respect to  $k$ .

**Proof Ideas for the Lower Bounds.** We only consider the non-universality problem here. Full proof details can be found in the full version of this paper [20]. Theorem 6 can be proven by a reduction from CNF-SAT. In order to get the ETH and SETH lower bounds, this reduction must yield instances with a  $(k-1)$ -tuple of gap constraints, where  $k$  is the number of Boolean variables. Theorem 8 can be shown by a similar reduction that starts from the standard parameterisation of the independent set problem. Both reductions can be defined by using a *meta non-universality problem* (METANUNI for short) as an intermediate step, which we define next.

Let  $\Gamma = \{b_1, b_2, \dots, b_m\}$  be some alphabet, and let  $q, k \in \mathbb{N}$ . An instance of the problem is a  $(q \times k)$ -matrix with the entries  $W_{i,j}$ , which are subsets of  $\Gamma$ . For every  $i \in [q]$ , we associate with row  $i$  of the matrix the language  $\mathcal{L}(W_i) = W_{i,1} \cdot W_{i,2} \cdots W_{i,k}$ , i. e., we simply represent the elements of  $W_{i,1} \times W_{i,2} \times \dots \times W_{i,k}$  as length- $k$  strings over  $\Gamma$  in the natural way. The question is then to decide whether  $\cup_{i \in [q]} \mathcal{L}(W_i) \neq \Gamma^k$  (see Figure 2 for an example).

We next discuss, how we can reduce CNF-SAT to METANUNI. Let  $F = \{c_1, c_2, \dots, c_q\}$  be a Boolean formula in CNF on variables  $\{v_1, \dots, v_k\}$  (i. e.,  $c_i \subseteq \{v_1, \neg v_1, \dots, v_k, \neg v_k\}$ ). We define alphabet  $\Gamma = \{0, 1\}$  and the  $(q \times k)$ -matrix with the entries  $W_{i,j}$  as follows. For every  $i \in [q]$  and  $j \in [k]$ , we define  $W_{i,j} = \{0\}$ , if  $v_j \in c_i$ ,  $W_{i,j} = \{1\}$ , if  $\neg v_j \in c_i$ , and  $W_{i,j} = \{0, 1\}$ , if  $\{v_j, \neg v_j\} \cap c_i = \emptyset$ . It can be verified with moderate effort, that for every  $i \in [q]$ ,  $\mathcal{L}(W_i)$  contains exactly the Boolean assignments that do not satisfy clause  $c_i$ . Hence,  $\cup_{i \in [q]} \mathcal{L}(W_i) \neq \{0, 1\}^k$  if and only if  $F$  is satisfiable (see Figure 2 for an example).

In a rather similar way, we can also phrase the independent set problem in terms of METANUNI. For the independent set problem, we get an undirected graph  $G = (V, E)$  with  $|V| = n$  and  $E = \{e_1, e_2, \dots, e_m\}$ , and a  $k \in [|V|]$ , and the question is whether  $G$  has a  $k$ -independent set, i. e., a set  $A \subseteq V$  with  $|A| = k$  and  $\{u, u'\} \notin E$  for every  $u, u' \in A$  with  $u \neq u'$ . This can be expressed in terms of METANUNI as follows. We interpret the set  $V$  of vertices as the alphabet  $\Gamma$ . We fix some bijection  $\nu : \{(i, r, s) \in [m] \times [k] \times [k] \mid r \neq s\} \rightarrow [mk(k-1)]$ . For every  $i \in [m]$  with  $e_i = (u, v)$ , and every  $r, s, j \in [k]$  with  $r \neq s$ , we define  $W_{\nu(i,r,s),j} = \{u\}$ , if  $j = r$ ,  $W_{\nu(i,r,s),j} = \{v\}$ , if  $j = s$ , and  $W_{\nu(i,r,s),j} = V$ , else. For example, if  $e_9 = (v_3, v_7)$  and  $k = 4$ , then row  $\nu(9, 2, 4)$  of the matrix would be  $V \{v_3\} V \{v_7\}$ .

It is a bit more difficult to see why this reduction works. The idea is that we represent sets of vertices of cardinality *at most*  $k$  by length- $k$  strings over  $V$  (note that sets of cardinality strictly less than  $k$  can be represented by strings with repeated symbols). For every edge  $(u, v)$  and for all pairs of positions  $r, s \in [k]$ , the language  $\mathcal{L}(W_{\nu(i,r,s)}) = W_{\nu(i,r,s),1}W_{\nu(i,r,s),2} \dots W_{\nu(i,r,s),k}$  represented by row  $\nu(i, r, s)$  of the matrix contains exactly the strings  $w \in \Gamma^k$  with  $(w[r], w[s]) = (u, v)$ , i. e., strings that represent non-independent sets with edge  $(u, v)$ . For the example  $e_9 = (v_3, v_7)$  and  $k = 4$ , we have  $\mathcal{L}(W_{\nu(9,2,4)}) = \{v_1v_3v_1v_7, v_2v_3v_1v_7, \dots, v_nv_3v_1v_7, \dots, v_1v_3v_2v_7, v_2v_3v_2v_7, \dots\}$ .

This whole idea works only because, in our setting, we assume that every vertex has a loop since then strings  $w$  of  $V^k$  contain an edge  $(w[r], w[s]) \in E$  for some  $r, s \in [k]$  if and only if the corresponding set of vertices is not independent or of cardinality strictly less than  $k$  (the latter is represented by a loop, i. e.,  $w[r] = w[s]$ ). In summary,  $G$  has a  $k$ -independent set if and only if not all length- $k$  strings are in  $\bigcup_{i \in [m], r, s \in [k], r \neq s} \mathcal{L}(W_{\nu(i,r,s)})$ .

The main technical challenge is to show a reduction from METANUNI to NUNI with length constraints. We next give a sketch of this reduction. Let  $\Gamma = \{b_1, b_2, \dots, b_m\}$ ,  $q, k \in \mathbb{N}$ , and, for every  $i \in [q], j \in [k]$ , let  $W_{i,j} \subseteq \Gamma$ . We transform this METANUNI instance into an instance of NUNI with length constraints as follows. We first define the alphabet  $\Sigma = \Gamma \cup \{\#\}$  (with  $\# \notin \Gamma$ ). Then we define a  $(k-1)$ -tuple  $gc = (C_1, C_2, \dots, C_{k-1})$  of gap constraints with  $C_i = (L^-(i), L^+(i)) = (m-1, 3m-1)$  for every  $i \in [k-1]$  (recall that  $m$  is  $\Gamma$ 's cardinality). To conclude the reduction, we have to construct a string  $K(W_1, \dots, W_q)$  over  $\Sigma$ , such that  $SubSeq(gc, K(W_1, \dots, W_q)) = \Sigma^k$  if and only if  $\bigcup_{i \in [q]} \mathcal{L}(W_i) = \Gamma^k$ . We do this in several steps.

For every  $i \in [q]$  and  $j \in [k]$ , let  $w_{i,j} \in \Gamma^*$  be some string representation of  $W_{i,j}$ , i. e.,  $\text{alph}(w_{i,j}) = W_{i,j}$  and  $|w_{i,j}| = |W_{i,j}| \leq m$ . For every  $i \in [q]$ , we define the string  $S(W_i) = w_{i,1}(\#)^{m-1}w_{i,2}(\#)^{m-1} \dots (\#)^{m-1}w_{i,k}$ .

We can show that those  $gc$ -subsequences of  $S(W_i)$  that do not contain occurrences of symbol  $\#$  must be mapped to  $S(W_i)$  in such a way that each  $j \in [k]$  is mapped to  $w_{i,j}$ . More precisely, for every  $i \in [q]$ , we have that  $(SubSeq(gc, S(W_i)) \cap \Gamma^*) = \mathcal{L}(W_i)$ . (†)

Next, we define a string  $T$  whose purpose it is to contain *all*  $gc$ -subsequences that contain at least one occurrence of  $\#$ . For every  $i \in [k]$ , let  $T_i = T_{i,1}T_{i,2} \dots T_{i,k}$ , where, for every  $j \in [k] \setminus \{i\}$ ,  $T_{i,j} = b_1b_2 \dots b_m\#^m$ , and  $T_{i,i} = \#^m$ . We define  $T$  by  $T = T_1(\#^{3m})T_2(\#^{3m}) \dots (\#^{3m})T_k$ . The idea here is that any  $gc$ -subsequence of  $T$  must be mapped entirely into some  $T_i$ , which, due to the length constraints, forces position  $i$  to be mapped to  $T_{i,i} = \#^m$ , i. e., to an occurrence of  $\#$ . More precisely, we have  $SubSeq(gc, T) = \{w \in \Sigma^k \mid |w|_{\#} \geq 1\}$ . (⊃)

Finally, we set  $K(W_1, \dots, W_q) = T(\#^{3m})S(W_1)(\#^{3m})S(W_2)(\#^{3m}) \dots (\#^{3m})S(W_q)$ . By using (†) and (⊃) from above, we can now prove  $SubSeq(gc, K(W_1, \dots, W_q)) = \Sigma^k$  if and only if  $\bigcup_{i \in [q]} \mathcal{L}(W_i) = \Gamma^k$ , which concludes the proof of correctness.

For proving Theorem 7, using METANUNI as an intermediate step seems not possible, since it introduces another symbol  $\#$  to the alphabet. However, we can devise a similar reduction. The main difference is that we represent each Boolean variable by two consecutive symbols of the subsequence, i. e., we need a  $(2k-1)$  tuple of length constraints (therefore,

the reduction does not yield a SETH bound as mentioned in Theorem 6). Since we cannot conveniently use a separator  $\#$  that is not used for expressing Boolean assignments, the constructed string is more complicated in this reduction (see the full version of this paper [20] for full details).

## 5 Special Variants

In the last section of this paper, we consider two natural variants of our setting, and show how the particularities of these variants have a substantial impact on the complexity of the problems investigated above.

**Gap Length Equalities.** We investigate whether the polynomiality of the matching problem (see Section 3) is preserved under adding *gap length equalities* to the gap constraints, i. e., constraints of the form  $|\mathbf{gap}_i| = |\mathbf{gap}_j|$  which are satisfied by an embedding  $e$  with respect to  $w$  if  $|\mathbf{gap}_e(w, i)| = |\mathbf{gap}_e(w, j)|$ . Our main motivation is that such length equality constraints (and more complex ones, e. g., described by linear inequalities like  $2|\mathbf{gap}_7| + |\mathbf{gap}_3| \leq |\mathbf{gap}_2|$ ) are of interest in the theory of string solving [3]. Unfortunately, the matching problem becomes immediately NP-hard (the following result can be shown by adapting the NP-completeness proof for matching patterns with variables from [4]; see the the full version of this paper [20] for details).

► **Theorem 9.** *MATCH with length constraints and gap length equalities is NP-complete, even for binary alphabets and length constraints  $(0, +\infty)$ .*

**Gap Constrained Subsequences With Multiplicities.** With respect to the equivalence problem, we can show a positive result for the following modified setting. Let us consider the  $gc$ -subsequences of the sets  $SubSeq(gc, w)$  with multiplicities. For example, for  $w_1 = \mathbf{a b b a}$  and  $w_2 = \mathbf{a b a b}$ , we have  $SubSeq(gc_2, w_1) = SubSeq(gc_2, w_2) = \{\mathbf{a a}, \mathbf{a b}, \mathbf{b a}, \mathbf{b b}\}$  with  $gc_2 = (\Sigma^*)$ . There is exactly one way of embedding  $\mathbf{a a}$  and  $\mathbf{b b}$  into both  $w_1$  and  $w_2$ . On the other hand,  $\mathbf{a b}$  can be embedded into  $w_1$  in two different ways and into  $w_2$  in three different ways. More precisely, the sets of  $gc_2$ -subsequences of  $w_1$  and  $w_2$  with multiplicities are  $\{(\mathbf{a a}, 1), (\mathbf{a b}, 2), (\mathbf{b a}, 2), (\mathbf{b b}, 1)\}$  and  $\{(\mathbf{a a}, 1), (\mathbf{a b}, 3), (\mathbf{b a}, 1), (\mathbf{b b}, 1)\}$ , respectively.

Let us now formalise this setting. For strings  $u$  and  $v$ , and a  $(|u| - 1)$ -tuple  $gc$  of gap constraints, we denote by  $|u|_{v, gc}$  the number of distinct embeddings  $e : |u| \rightarrow |v|$  that satisfy  $gc$  and  $v \preceq_e u$ . For example,  $|\mathbf{b b a a}|_{\mathbf{b a}, gc_2} = 4$ , as  $u[1]u[3] = u[2]u[3] = u[1]u[4] = u[2]u[4] = \mathbf{b a}$ . For any  $(k - 1)$ -tuple  $gc$  of gap constraints, we define the function  $\Psi_{gc}(\cdot) : \Sigma^* \rightarrow \mathbb{N}^{(\Sigma^k)}$  by  $\Psi_{gc}(w)[p] = |w|_{p, gc}$  for every  $p \in \Sigma^k$ . The *equivalence problem with multiplicities* is to decide, for a given  $(k - 1)$ -tuple  $gc$  of gap constraints, and strings  $w, w' \in \Sigma^*$ , whether  $\Psi_{gc}(w) = \Psi_{gc}(w')$ . Note that for the case  $gc = (\Sigma^*, \dots, \Sigma^*)$  this is called the  $k$ -binomial equivalence, and was studied in the area of combinatorics on words (see, e. g., [55, 47, 48, 26]).

We show that equivalence with multiplicities can be decided in polynomial time (in contrast to the NP-completeness of the case without multiplicities).

► **Theorem 10.** *If  $gc = (C_1, \dots, C_{k-1})$  and  $C_i$  can be decided in polynomial time then the equivalence problem with multiplicities can be solved in polynomial time.*

**Proof Sketch.** We adapt the main idea from [26]. For a given string  $w$  and tuple  $gc$  of gap constraints, we can construct in polynomial time an NFA  $A_{w, gc}$  with  $L(A_{w, gc}) = SubSeq(gc, w)$ . Moreover, for every  $p \in SubSeq(gc, w)$ , there is a one-to-one correspondence

between distinct embeddings  $e$  with  $p \preceq_e w$  satisfying  $gc$  and accepting paths of  $A_{w,gc}$  on input  $p$ . Consequently, we can decide  $\Psi_{gc}(w) = \Psi_{gc}(w')$  by deciding the *path equivalence* of  $A_{w,gc}$  and  $A_{w',gc}$ , which can be done in polynomial time by the algorithm from [64]. ◀

In the case of length, regular or reg-len constraints, the equivalence problem with multiplicities can be solved in  $O(\max\{|w|, |w'|\}^4 k^4 + \text{size}(gc))$  time. The *containment problem with multiplicities* (i. e., deciding  $\Psi_{gc}(w)[p] \leq \Psi_{gc}(w')[p]$  for all  $p \in \Sigma^k$ ) seems to be more difficult. To our knowledge, whether the case of classical subsequences (i. e., length constraints  $(0, \infty)$ ) can be solved in polynomial time is open. On the other hand, for the case of length constraints  $(0, 0)$  only (i. e., consecutive factors), or of length constraints  $(\ell, \ell)$  only (i. e., the case of the so called *partial words* [11, 18], where each gap has fixed length  $\ell$ , and can be seen as a factor of length  $\ell$  of wild cards, or don't care positions), showing polynomial time solvability is relatively simple.

---

## References

- 1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.
- 2 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 39–51, 2014. doi:10.1007/978-3-662-43948-7\_4.
- 3 Roberto Amadini. A survey on string constraint solving. *ACM Computing Surveys (CSUR)*, 55(1):1–38, 2021.
- 4 Dana Angluin. Finding patterns common to a set of strings. *J. Comput. Syst. Sci.*, 21(1):46–62, 1980.
- 5 Alexander Artikis, Alessandro Margara, Martín Ugarte, Stijn Vansummeren, and Matthias Weidlich. Complex event recognition languages: Tutorial. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*, pages 7–10, 2017. doi:10.1145/3093742.3095106.
- 6 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466, 2016. doi:10.1109/FOCS.2016.56.
- 7 Johannes Bader, Simon Gog, and Matthias Petri. Practical variable length gap pattern matching. In *Experimental Algorithms – 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings*, pages 1–16, 2016. doi:10.1007/978-3-319-38851-9\_1.
- 8 Ricardo A. Baeza-Yates. Searching subsequences. *Theor. Comput. Sci.*, 78(2):363–376, 1991.
- 9 Gabriel Bathie and Tatiana Starikovskaya. Property testing of regular languages with applications to streaming property testing of visibly pushdown languages. In *ICALP*, volume 198 of *LIPICs*, pages 119:1–119:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 10 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and David Kofoed Wind. String matching with variable length gaps. *Theor. Comput. Sci.*, 443:25–34, 2012. doi:10.1016/j.tcs.2012.03.029.
- 11 Francine Blanchet-Sadri. *Algorithmic Combinatorics on Partial Words*. Discrete mathematics and its applications. CRC Press, 2008.
- 12 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly sub-quadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

- 13 Karl Bringmann. Fine-grained complexity theory (tutorial). In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 4:1–4:7, 2019. doi:10.4230/LIPIcs.STACS.2019.4.
- 14 Karl Bringmann and Bhaskar Ray Chaudhury. Sketching, streaming, and fine-grained complexity of (weighted) LCS. In *Proc. FSTTCS 2018*, volume 122 of *LIPIcs*, pages 40:1–40:16, 2018.
- 15 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proc. SODA 2018*, pages 1216–1235, 2018.
- 16 Sam Buss and Michael Soltys. Unshuffling a square is NP-hard. *J. Comput. Syst. Sci.*, 80(4):766–776, 2014. doi:10.1016/j.jcss.2013.11.002.
- 17 Manuel Cáceres, Simon J. Puglisi, and Bella Zhukova. Fast indexes for gapped pattern matching. In *SOFSEM 2020: Theory and Practice of Computer Science – 46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus, January 20-24, 2020, Proceedings*, pages 493–504, 2020. doi:10.1007/978-3-030-38919-2\_40.
- 18 Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Inf. Process. Lett.*, 101(2):53–54, 2007.
- 19 Maxime Crochemore, Borivoj Melichar, and Zdenek Troníček. Directed acyclic subsequence graph – Overview. *J. Discrete Algorithms*, 1(3-4):255–280, 2003.
- 20 Joel D. Day, Maria Kosche, Florin Manea, and Markus L. Schmid. Subsequences with gap constraints: Complexity bounds for matching and analysis problems. *CoRR*, 2022. doi:10.48550/ARXIV.2206.13896.
- 21 Patrick Dinklage, Johannes Fischer, Alexander Herlez, Tomasz Kociumaka, and Florian Kurpicz. Practical Performance of Space Efficient Data Structures for Longest Common Extensions. In *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:20, 2020.
- 22 Bartłomiej Dudek, Pawel Gawrychowski, Garance Gourdel, and Tatiana Starikovskaya. Streaming regular expression membership and pattern matching. In *SODA*, pages 670–694. SIAM, 2022.
- 23 Henning Fernau, Florin Manea, Robert Mercas, and Markus L. Schmid. Pattern matching with variables: Efficient algorithms and complexity results. *ACM Trans. Comput. Theory*, 12(1):6:1–6:37, 2020.
- 24 Lukas Fleischer and Manfred Kufleitner. Testing Simon’s congruence. In *Proc. MFCS 2018*, volume 117 of *LIPIcs*, pages 62:1–62:13, 2018.
- 25 Dominik D. Freydenberger, Pawel Gawrychowski, Juhani Karhumäki, Florin Manea, and Wojciech Rytter. Testing  $k$ -binomial equivalence. In *Multidisciplinary Creativity, a collection of papers dedicated to G. Păun 65th birthday*, pages 239–248, 2015. available in CoRR abs/1509.00622.
- 26 Dominik D Freydenberger, Pawel Gawrychowski, Juhani Karhumäki, Florin Manea, and Wojciech Rytter. Testing  $k$ -binomial equivalence. *arXiv preprint arXiv:1509.00622*, 2015.
- 27 Moses Ganardi, Danny Hucce, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *STACS*, volume 96 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 28 Moses Ganardi, Danny Hucce, and Markus Lohrey. Querying regular languages over sliding windows. In *FSTTCS*, volume 65 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.
- 29 Moses Ganardi, Danny Hucce, and Markus Lohrey. Randomized sliding window algorithms for regular languages. In *ICALP*, volume 107 of *LIPIcs*, pages 127:1–127:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 30 Moses Ganardi, Danny Hucce, and Markus Lohrey. Sliding window algorithms for regular languages. In *LATA*, volume 10792 of *Lecture Notes in Computer Science*, pages 26–35. Springer, 2018.



- 31 Moses Ganardi, Danny Hucke, Markus Lohrey, and Tatiana Starikovskaya. Sliding window property testing for regular languages. In *ISAAC*, volume 149 of *LIPICs*, pages 6:1–6:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 32 Emmanuelle Garel. Minimal separators of two words. In *Proc. CPM 1993*, volume 684 of *Lecture Notes in Computer Science*, pages 35–53, 1993.
- 33 Pawel Gawrychowski, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Efficiently testing Simon’s congruence. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, pages 34:1–34:18, 2021. doi:10.4230/LIPICs.STACS.2021.34.
- 34 Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. Complex event recognition in the big data era: a survey. *VLDB J.*, 29(1):313–352, 2020. doi:10.1007/s00778-019-00557-w.
- 35 Simon Halfon, Philippe Schnoebelen, and Georg Zetsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *Proc. LICS 2017*, pages 1–12, 2017.
- 36 Jean-Jacques Hebrard. An algorithm for distinguishing efficiently bit-strings by their subsequences. *Theor. Comput. Sci.*, 82(1):35–49, 22 May 1991.
- 37 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
- 38 Costas S. Iliopoulos, Marcin Kubica, M. Sohel Rahman, and Tomasz Walen. Algorithms for computing the longest parameterized common subsequence. In *Combinatorial Pattern Matching, 18th Annual Symposium, CPM 2007, London, Canada, July 9-11, 2007, Proceedings*, pages 265–273, 2007. doi:10.1007/978-3-540-73437-6\_27.
- 39 Russell Impagliazzo and Ramamohan Paturi. On the complexity of  $k$ -SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 40 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- 41 Prateek Karandikar, Manfred Kufleitner, and Philippe Schnoebelen. On the index of Simon’s congruence for piecewise testability. *Inf. Process. Lett.*, 115(4):515–519, 2015.
- 42 Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages with applications in logical complexity. In *Proc. CSL 2016*, volume 62 of *LIPICs*, pages 37:1–37:22, 2016.
- 43 Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages and the complexity of the logic of subwords. *Log. Methods Comput. Sci.*, 15(2), 2019.
- 44 Sarah Kleest-Meißner, Rebecca Sattler, Markus L. Schmid, Nicole Schweikardt, and Matthias Weidlich. Discovering event queries from traces: Laying foundations for subsequence-queries with wildcards and gap-size constraints. In *25th International Conference on Database Theory, ICDT 2022*, volume 220 of *LIPICs*, pages 18:1–18:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICDT.2022.18.
- 45 Dietrich Kuske. The subtrace order and counting first-order logic. In *Proc. CSR 2020*, volume 12159 of *Lecture Notes in Computer Science*, pages 289–302, 2020.
- 46 Dietrich Kuske and Georg Zetsche. Languages ordered by the subword order. In *Proc. FOSSACS 2019*, volume 11425 of *Lecture Notes in Computer Science*, pages 348–364, 2019.
- 47 Marie Lejeune, Julien Leroy, and Michel Rigo. Computing the  $k$ -binomial complexity of the Thue-Morse word. In *Proc. DLT 2019*, volume 11647 of *Lecture Notes in Computer Science*, pages 278–291, 2019.
- 48 Julien Leroy, Michel Rigo, and Manon Stipulanti. Generalized Pascal triangle for binomial coefficients of words. *Electron. J. Combin.*, 24(1.44):36 pp., 2017.
- 49 Chun Li and Jianyong Wang. Efficiently mining closed subsequences with gap constraints. In *SDM*, pages 313–322. SIAM, 2008.

- 50 Chun Li, Qingyan Yang, Jianyong Wang, and Ming Li. Efficient mining of gap-constrained subsequences and its various applications. *ACM Trans. Knowl. Discov. Data*, 6(1):2:1–2:39, 2012.
- 51 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bull. EATCS*, 105:41–72, 2011. URL: <http://eatcs.org/beatcs/index.php/beatcs/article/view/92>.
- 52 David Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, April 1978.
- 53 Alexandru Mateescu, Arto Salomaa, and Sheng Yu. Subword histories and Parikh matrices. *J. Comput. Syst. Sci.*, 68(1):1–21, 2004.
- 54 William E. Riddle. An approach to software system modelling and analysis. *Comput. Lang.*, 4(1):49–66, 1979. doi:10.1016/0096-0551(79)90009-2.
- 55 Michel Rigo and Pavel Salimov. Another generalization of abelian equivalence: Binomial complexity of infinite words. *Theor. Comput. Sci.*, 601:47–57, 2015.
- 56 Arto Salomaa. Connections between subwords and certain matrix mappings. *Theoret. Comput. Sci.*, 340(2):188–203, 2005.
- 57 Shinnosuke Seki. Absoluteness of subword inequality is undecidable. *Theor. Comput. Sci.*, 418:116–120, 2012. doi:10.1016/j.tcs.2011.10.017.
- 58 Alan C. Shaw. Software descriptions with flow expressions. *IEEE Trans. Software Eng.*, 4(3):242–254, 1978. doi:10.1109/TSE.1978.231501.
- 59 Imre Simon. *Hierarchies of events with dot-depth one — Ph.D. thesis*. University of Waterloo, 1972.
- 60 Imre Simon. Piecewise testable events. In *Autom. Theor. Form. Lang., 2nd GI Conf.*, volume 33 of *LNCS*, pages 214–222, 1975.
- 61 Imre Simon. Words distinguished by their subwords (extended abstract). In *Proc. WORDS 2003*, volume 27 of *TUCS General Publication*, pages 6–13, 2003.
- 62 Ken Thompson. Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968. doi:10.1145/363347.363387.
- 63 Zdenek Troníček. Common subsequence automaton. In *Proc. CIAA 2002 (Revised Papers)*, volume 2608 of *Lecture Notes in Computer Science*, pages 270–275, 2002.
- 64 Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM J. Comput.*, 21(2):216–227, 1992. doi:10.1137/0221017.
- 65 Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 17–29, 2015. doi:10.4230/LIPIcs.IPEC.2015.17.
- 66 Georg Zetsche. The complexity of downward closure comparisons. In *Proc. ICALP 2016*, volume 55 of *LIPIcs*, pages 123:1–123:14, 2016.
- 67 Haopeng Zhang, Yanlei Diao, and Neil Immerman. On complexity and optimization of expensive queries in complex event processing. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 217–228, 2014. doi:10.1145/2588555.2593671.