


Subsequences with Generalised Gap Constraints: Upper and Lower Complexity Bounds

Florin Manea ✉ 

Computer Science Department and CIDAS, Universität Göttingen, Germany

Jonas Richardsen ✉

Computer Science Department and CIDAS, Universität Göttingen, Germany

Markus L. Schmid ✉ 

Humboldt-Universität zu Berlin, Berlin, Germany

Abstract

For two strings u, v over some alphabet A , we investigate the problem of embedding u into w as a subsequence under the presence of generalised gap constraints. A generalised gap constraint is a triple $(i, j, C_{i,j})$, where $1 \leq i < j \leq |u|$ and $C_{i,j} \subseteq A^*$. Embedding u as a subsequence into v such that $(i, j, C_{i,j})$ is satisfied means that if $u[i]$ and $u[j]$ are mapped to $v[k]$ and $v[\ell]$, respectively, then the induced gap $v[k+1..l-1]$ must be a string from $C_{i,j}$. This generalises the setting recently investigated in [Day et al., ISAAC 2022], where only gap constraints of the form $C_{i,i+1}$ are considered, as well as the setting from [Kosche et al., RP 2022], where only gap constraints of the form $C_{1,|u|}$ are considered.

We show that subsequence matching under generalised gap constraints is NP-hard, and we complement this general lower bound with a thorough (parameterised) complexity analysis. Moreover, we identify several efficiently solvable subclasses that result from restricting the interval structure induced by the generalised gap constraints.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms; Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Formal languages and automata theory

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.CPM.2024.22

Funding *Florin Manea*: Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) in the Heisenberg programme, project number 466789228.

Markus L. Schmid: Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) – project number 522576760 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 522576760).

1 Introduction

For a string $v = v_1v_2 \dots v_n$, where each v_i is a single symbol from some alphabet Σ , any string $u = v_{i_1}v_{i_2} \dots v_{i_k}$ with $k \leq n$ and $1 \leq i_1 < i_2 < \dots < i_k \leq n$ 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 $\mathbf{a\ b\ a\ c\ b\ b\ a}$ has among its subsequences $\mathbf{a\ a\ a}$, $\mathbf{a\ b\ c\ a}$, $\mathbf{c\ b\ a}$, and $\mathbf{a\ b\ a\ b\ b\ a}$. With respect to $\mathbf{a\ a\ a}$, there exists just one embedding, namely $1 \mapsto 1$, $2 \mapsto 3$, and $3 \mapsto 7$, but there are two embeddings for $\mathbf{c\ b\ a}$.

This classical concept of subsequences is employed in many different areas of computer science: in formal languages and logics (e. g., piecewise testable languages [54, 55, 29, 30, 31, 47], or subword order and downward closures [26, 38, 37, 59]), in combinatorics on



© Florin Manea, Jonas Richardsen, and Markus L. Schmid;
licensed under Creative Commons License CC-BY 4.0

35th Annual Symposium on Combinatorial Pattern Matching (CPM 2024).

Editors: Shunsuke Inenaga and Simon J. Puglisi; Article No. 22; pp. 22:1–22:17

Leibniz International Proceedings in Informatics



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

words [49, 23, 40, 39, 52, 44, 50, 51], for modelling concurrency [48, 53, 12], in database theory (especially event stream processing [4, 25, 60, 33, 34, 24]). Moreover, many classical algorithmic problems are based on subsequences, e. g., longest common subsequence [6] or shortest common supersequence [43] (see [3, 22] and the survey [36], for recent results on string problems concerned with subsequences). Note that the longest common subsequence problem, in particular, has recently regained substantial interest in the context of fine-grained complexity (see [10, 11, 1, 2]).

In this paper, we are concerned with the following special setting of subsequences recently introduced in [17]. If a string u is a subsequence of a string v via an embedding e , then this embedding e also induces $|u| - 1$ so-called *gaps*, i. e., the (possibly empty) factors $v_{e(i)+1}v_{e(i)+2} \dots v_{e(i+1)-1}$ of v that lie strictly between the symbols where u is mapped to. For example, $\mathbf{a c b} \preceq_e \mathbf{a b a c b b a}$ with e being defined by $1 \mapsto 1$, $2 \mapsto 4$, and $3 \mapsto 6$ induces the gaps $\mathbf{b a}$ and \mathbf{b} . We can now restrict the subsequence relation by adding *gap constraints* as follows. A string u is accompanied by $|u| - 1$ gap constraints $C_1, C_2, \dots, C_{|u|-1} \subseteq \Sigma^*$, and u is a valid subsequence of a string v under these gap constraints, if $u \preceq_e v$ for an embedding e that induces gaps from the gap constraints, i. e., the i^{th} gap is in C_i .

Such gap-constrained subsequences allow to model situations for which classical subsequences are not expressive enough. For example, if we model concurrency by shuffling together strings that represent threads on a single processor, then fairness properties of a scheduler usually imply that the gaps of these subsequences are not huge. Or assume that we compute an alignment between two strings by computing a long common subsequence. Then it is not desirable 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. In fact, an overall shorter common subsequence that does not contain such huge gaps seems to induce a more reasonable alignment (this setting is investigated in [3]). Another example is complex event processing: Assume that a log-file contains a sequence of events of the run of a large system. Then we might query this string for the situation that between some 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). This can be modeled by 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 (such subsequence queries are investigated in [33, 34, 24]).

In [17], two types of gap constraints are considered: Length constraints $C = \{w \in \Sigma^* \mid \ell \leq |w| \leq k\}$, and regular constraints where C is just a regular language over Σ^* , as well as combinations of both. In a related paper, [35], the authors went in a slightly different direction, and were interested in subsequences appearing in bounded ranges, which is equivalent to constraining the length of the string occurring between the first and last symbol of the embedding, namely $v_{e(1)+1}v_{e(i)+2} \dots v_{e(m)-1}$. In this paper, we follow up on the work of [17, 35], but significantly generalise the concept of gap constraints. Assume that $u \preceq_e v$. Instead of only considering the gaps given by the images of two consecutive positions of u , we consider each string $v_{e(i)+1}v_{e(i)+2} \dots v_{e(j)-1}$ of v as a gap, where $i, j \in \{1, 2, \dots, |u|\}$ with $i < j$ (note that these general gaps also might contain symbols from v that correspond to images of e , namely $e(i+1), e(i+2), \dots, e(j-1)$). For example, $\mathbf{a b a c} \preceq_e \mathbf{b a a b b c a c c a b}$ with e defined by $1 \mapsto 2$, $2 \mapsto 5$, $3 \mapsto 7$ and $4 \mapsto 9$ induces the following gaps: The (1, 2)-gap $\mathbf{a b}$, the (2, 3)-gap \mathbf{c} , the (3, 4)-gap \mathbf{c} , the (1, 3)-gap $\mathbf{a b b c}$, the (2, 4)-gap $\mathbf{c a c}$, and the (1, 4)-gap $\mathbf{a b b c a c}$. In this more general setting, we can now add gap-constraints in an analogous way as before. For example, the gap constraint $C_{2,4} = \{\mathbf{a}, \mathbf{c}\}^*$ for the (2, 4)-gap, the gap constraint $C_{1,4} = \{w \in \Sigma^* \mid 3 \leq |w| \leq 5\}$ for the (1, 4)-gap and the gap constraint

$C_{2,3} = \{c^n \mid n \geq 1\}$ for the (2,3)-gap. Under these gap-constraints, the embedding e defined above is not valid: The gap constraints $C_{2,4}$ and $C_{2,3}$ are satisfied, but the (1,4)-gap $a b b c a c$ is too long for gap constraint $C_{1,4}$. However, changing $4 \mapsto 9$ into $4 \mapsto 8$ yields an embedding that satisfies all gap constraints.

Our Contribution. We provide an in-depth analysis of the complexity of the *matching problem* associated with the setting explained above, i.e., for given strings u, v and a set \mathcal{C} of generalised gap-constraints for u , decide whether or not $u \preceq_e v$ for an embedding e that satisfies all constraints in \mathcal{C} . We concentrate on two different kinds of constraints: semilinear constraints of the form $\{w \in \Sigma^* \mid |w| \in S\}$, where S is a semilinear set, and regular constraints.

In general, this matching problem is NP-complete for both types of constraints (demonstrating a stark contrast to the simpler setting of gap constraints investigated in [17, 35]), and this even holds for binary alphabets and if each semilinear constraint has constant size, and also if every regular constraint is represented by an automaton with a constant number of states. On the other hand, if the number of constraints is bounded by a constant, then the matching problem is solvable in polynomial-time, but, unfortunately, we obtain W[1]-hardness even if the complete size $|u|$ is a parameter (also for both types of constraints). An interesting difference in complexity between the two types of constraints is pointed out by the fact that for regular constraints the matching problem is fixed-parameter tractable if parameterised by $|u|$ and the maximum size of the regular constraints (measured in the size of a DFA), while for semilinear constraints this variant stays W[1]-hard.

We then show that structurally restricting the interval structure induced by the given constraints yields polynomial-time solvable subclasses. Moreover, if the interval structure is completely non-intersecting, then we obtain an interesting subcase for which the matching problem can be solved in time $O(n^\omega |\mathcal{C}|)$, where $O(n^\omega)$ is the time needed to multiply two $n \times n$ Boolean matrices. We complement this result by showing that an algorithm with running time $\mathcal{O}(|w|^g |\mathcal{C}|^h)$ with $g + h < 3$ would refute the strong exponential time hypothesis. While this is not a tight lower bound, we wish to point out that, due to the form of our algorithm, which boils down to performing $O(|\mathcal{C}|)$ matrix multiplications, a polynomially stronger lower bound would have proven that matrix multiplication in quadratic time is not possible.

Related Work. Our work extends [17, 35]. However, subsequences with various types of gap constraints have been considered before, mainly in the field of combinatorial pattern matching with biological motivations (see [7, 41, 42, 27] and [5, 13] for more practical papers).

2 Preliminaries

Let $\mathbb{N} = \{1, 2, \dots\}$, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. For $m, n \in \mathbb{N}_0$ let $[m, n] = \{k \in \mathbb{N}_0 \mid m \leq k \leq n\} = \{m, \dots, n\}$ and $[n] = [1, n]$. For some alphabet Σ and some length $n \in \mathbb{N}_0$ we define Σ^n as the set of all words of length n over Σ (with Σ^0 only containing the empty word ε). Furthermore $\Sigma^* := \bigcup_{n \in \mathbb{N}_0} \Sigma^n$ is the set of all words over Σ . For some $w \in \Sigma^*$, $|w|$ is the length of w , $w[i]$ denotes the i -th character of w and $w[i..j] := w[i] \dots w[j]$ is the substring of w from the i -th to the j -th character (where $i, j \in [|w|], i \leq j$).

We use deterministic and nondeterministic finite automata (DFA and NFA) as commonly defined in the literature; as a particularity, for the sake of having succinct representations of automata, we allow DFAs to be incomplete: given a state q of a DFA and a letter a , the

transition from q with a may be left undefined, which means that the computations of the DFA on the inputs which lead to the respective transition are not-accepting. For a DFA or NFA A , we denote by $\text{size}(A)$ its total size, and by $\text{states}(A)$ its number of states. Note that if A is a DFA over alphabet Σ , then we have that $\text{size}(A) = O(\text{states}(A)|\Sigma|)$.

A subset $L \subseteq \mathbb{N}$ is called *linear*, if there are $m \in \mathbb{N}_0$ and $x_0 \in \mathbb{N}_0, x_1, \dots, x_m \in \mathbb{N}$, such that $L = L(x_0; x_1, \dots, x_m) := \{x_0 + \sum_{i=1}^m k_i x_i \mid k_1, \dots, k_m \in \mathbb{N}_0\}$. For $m = 0$, we write $L(x_0) = \{x_0\}$. We can assume without loss of generality that $x_i \neq x_j$ for $i \neq j, i, j \in [m]$. A set S is *semilinear*, if it is a finite union of linear sets (see also [46]).

We assume that each integer involved in the representation of a linear set fits into constant memory (see our discussion about the computational model at the end of this section). Consequently, we measure the size of a linear set $L = L(x_0; x_1, \dots, x_m)$ as $\text{size}(L) = m + 1$, and the size of a semilinear set $S = L_1 \cup L_2 \cup \dots \cup L_k$ is measured as $\text{size}(S) = \sum_{i=1}^k \text{size}(L_i)$. In other words, $\text{size}(S)$ is the number of integers used for defining S .

Computational Model. For the complexity analysis of the algorithmic problems described in this paper we assume the *unit-cost RAM model with logarithmic word size* (see [15]). This means that for input size N , the memory words of the model can store $\log N$ bits. Thus, if we have input words of length N , they are over an alphabet which has at most $\sigma \leq N$ different characters, which we can represent using the *integer alphabet* $\Sigma = [\sigma]$. As such, we can store each character within one word of the model. Then, it is possible to read, write and compare single characters in one unit time.

Complexity Hypotheses. Let us consider the *Satisfiability problem for formulas in conjunctive normal form*, or CNF-SAT for short. Here, given a boolean formula F in conjunctive normal form, i.e., $F = \{c_1, \dots, c_m\}$ and $c_i \subseteq \{v_1, \dots, v_n, \neg v_1, \dots, \neg v_n\}$ for variables v_1, \dots, v_n , it is to be determined whether F is satisfiable. This problem was shown to be NP-hard [14]. By restricting $|c_i| \leq k$ for all $i \in [m]$ we obtain the problem of k -CNF-SAT. We will base our lower bound on the following algorithmic hypothesis:

► **Hypothesis 1** (Strong Exponential Time Hypothesis (SETH) [28]). *For any $\varepsilon > 0$, there exists a $k \in \mathbb{N}$, such that k -CNF-SAT cannot be solved in $\mathcal{O}(2^{n(1-\varepsilon)}) \text{poly}(m)$ time, where $\text{poly}(n)$ is an arbitrary (but fixed) polynomial function.*

The *Clique problem*, CLIQUE, asks, given a graph G and a number $k \in \mathbb{N}$, whether G has a k -clique. Hereby, a k -clique is a subset of k pairwise adjacent vertices, i.e., there is an edge between any pair of vertices in the subset. Since CNF-SAT can be reduced to CLIQUE [32], the latter is also NP-hard.

The *k -Orthogonal Vectors problem*, k -OV, receives as inputs k sets V_1, \dots, V_k each containing n elements from $\{0, 1\}^d$ for some $d \in \omega(\log n)$, i.e., d -dimensional boolean vectors. The question is, whether one can select vectors $\vec{v}_i \in V_i$ for $i \in [k]$ such that the vectors are orthogonal: $\sum_{j=1}^n \prod_{i=1}^k \vec{v}_i[j] = 0$. It is possible to show the following lemma ([58, 57]):

► **Lemma 2.** *k -OV cannot be solved in $n^{k-\varepsilon} \text{poly}(d)$ time for any $\varepsilon > 0$, unless SETH fails.*

This lemma will later form the basis for the conditional lower bound in the case of non-intersecting constraints.

3 Subsequences with Gap Constraints

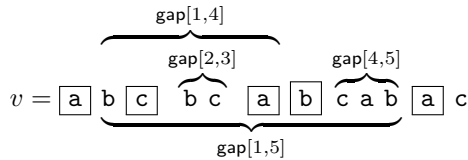
An *embedding* is any function $e : [k] \rightarrow [\ell]$ for some $k, \ell \in \mathbb{N}$ with $k \leq \ell$, such that $e(1) < e(2) < \dots < e(k)$ (note that this also implies that $1 \leq e(1)$ and $e(k) \leq \ell$). Let Σ be some alphabet. For a string $v = v_1 v_2 \dots v_n$, where $v_i \in \Sigma$ for every $i \in [n]$, any

string $u = v_{i_1}v_{i_2} \dots v_{i_k}$ with $k \leq n$ and $1 \leq i_1 < i_2 < \dots < i_k \leq n$ is called a *subsequence* (or, alternatively, *scattered factor* or *subword*) of v (denoted by $u \preceq v$). Every embedding $e : [k] \rightarrow [|v|]$ with $k \leq |v|$ induces the subsequence $u_e = v_{e(1)}v_{e(2)} \dots v_{e(k)}$ of v . If u is a subsequence of v induced by an embedding e , then we denote this by $u \preceq_e v$; we also say that an embedding e witnesses $u \preceq v$ if $u \preceq_e v$. When embedding a substring $u[s..t]$ for some $s, t \in [|u|], s \leq t$, we use a partial embedding $e : [s, t] \rightarrow [n]$ and write $u[s..t] \preceq_e w$.

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 different embeddings for **c b a**.

Let $v \in \Sigma^*$ and let $e : [k] \rightarrow [|v|]$ be an embedding. For every $i, j \in [k]$ with $i < j$, the string v and the embedding e induces the (i, j) -gap, which is the factor of v that occurs strictly between the positions corresponding to the images of i and j under the embedding e , i. e., $\text{gap}_{v,e}[i, j] = v[e(i) + 1..e(j) - 1]$. If v and e are clear from the context, we also drop this dependency in our notation, i. e., we also write $\text{gap}[i, j]$.

As an example, consider $v = \mathbf{a b c b c a b c a b a c}$ and $u = \mathbf{a c a b a}$. There are several embeddings $e : [|u|] \rightarrow [|v|]$ that witness $u \preceq v$. Each such embedding also induces an (i, j) -gap for every $i, j \in [5]$ with $i < j$. For the embedding e with $e(1) = 1, e(2) = 3, e(3) = 6, e(4) = 7, e(5) = 11$ (that satisfies $u \preceq_e v$), some of these gaps are illustrated below (note that e is also indicated by the boxed symbols of v):



A *gap-constraint* for a string $u \in \Sigma^*$ is a triple $C = (i, j, L)$ with $1 \leq i < j \leq |u|$ and $L \subseteq \Sigma^*$. A gap-constraint $C = (i, j, L)$ is also called an (i, j) -gap-constraint. The component L is also called the *gap-constraint language* of the gap-constraint (i, j, L) . We say that a string v and some embedding $e : [|u|] \rightarrow [|v|]$ satisfies the gap-constraint C if and only if $\text{gap}_{v,e}[i, j] \in L$.

As an example, let us define some gap-constraints for the string $u = \mathbf{a c a b a}$: $(1, 4, \Sigma^*)$, $(1, 5, \{w_1 c w_2 c w_3 \mid w_1, w_2, w_3 \in \Sigma^*\})$, $(2, 3, \{w \in \Sigma^* \mid |w| \geq 5\})$ and $(4, 5, \{w \in \Sigma^* \mid |w| \leq 4\})$. It can be easily verified that the gaps induced by the string v and the embedding e defined above (and illustrated by the figure) satisfy all of these gap constraints, except $(2, 3, \{w \in \Sigma^* \mid |w| \geq 5\})$ since $|\text{gap}_{v,e}[2, 3]| = 2 < 5$. However, the embedding e' defined by $e'(1) = 1, e'(2) = 3, e'(3) = 9, e'(4) = 10$ and $e'(5) = 11$ is such that v and e' satisfy all of the mentioned gap-constraints (in particular, note that $|\text{gap}_{v,e'}[2, 3]| = 5$ and that $\text{gap}_{v,e'}[4, 5] = \varepsilon$).

A *set of gap-constraints* for u is a set \mathcal{C} that, for every $i, j \in \mathbb{N}$ with $i < j$, may contain at most one (i, j) -gap-constraint for u . A string v and some embedding $e : [|u|] \rightarrow [|v|]$ satisfy \mathcal{C} if v and e satisfy every gap-constraint of \mathcal{C} .

For strings $u, v \in \Sigma^*$ with $|u| \leq |v|$, and a set \mathcal{C} of gap-constraints for u , we say that u is a \mathcal{C} -subsequence of v , if $u \preceq_e v$ for some embedding e such that v and e satisfy \mathcal{C} . We shall also write $u \preceq_{\mathcal{C}} v$ to denote that u is a \mathcal{C} -subsequence of v .

The Matching Problem. The central decision problem that we investigate in this work is the following matching problem, MATCH, for subsequences with gap-constraints:

- *Input:* Two strings $w \in \Sigma^*$ (also called text), with $|w| = n$, and $p \in \Sigma^*$ (also called pattern), with $|p| = m \leq n$, and a non-empty set \mathcal{C} of gap-constraints.
- *Question:* Is p a \mathcal{C} -subsequence of w ?

Obviously, for this matching problem it is vital how we represent gap-constraints (i, j, L) , especially the gap-constraint language L . Moreover, since every possible language membership problem “ $v \in L?$ ” can be expressed as the matching problem instance $w = \#v\#, p = \#\#$ and $\mathcal{C} = (1, 2, L)$, where $\# \notin \Sigma$, we clearly should restrict our setting to constraints (i, j, L) with sufficiently simple languages L . These issues are discussed next.

Types of Gap-Constraints. A gap-constraint $C = (i, j, L)$ (for some string) is a

- *regular constraint* if $L \in \text{REG}$. We represent the gap-constraint language of a regular constraint by a deterministic finite automaton (for short, DFA). In particular, $\text{size}(C) = \text{size}(L) = \text{size}(A)$ and $\text{states}(C) = \text{states}(L) = \text{states}(A)$.
- *semilinear length constraint* if there is a semi-linear set S , such that $L = \{w \in \Sigma^* \mid |w| \in S\}$. We represent the gap-constraint language of a semi-linear length constraint succinctly by representing the semilinear set S in a concise way (i. e., as numbers in binary encoding). In particular, $\text{size}(C) = \text{size}(L) = \text{size}(S)$.

For a set \mathcal{C} of gap constraints, let $\text{size}(\mathcal{C}) = \sum_{C \in \mathcal{C}} \text{size}(C)$ and $\text{gapsize}(\mathcal{C}) = \max\{\text{size}(C) \mid C \in \mathcal{C}\}$. We have $\text{size}(\mathcal{C}) \leq |\mathcal{C}| \text{gapsize}(\mathcal{C})$.

Obviously, $\{v \in \Sigma^* \mid |v| \in S\}$ is regular for any semilinear set S . However, due to our concise representation, transforming a simple length constraint into a semilinear length constraints, or transforming a semilinear length constraint into a DFA representation may cause an exponential size increase.

By $\text{MATCH}_{\text{REG}}$ and $\text{MATCH}_{\text{SLS}}$ we denote the problem MATCH , where all gap constraints are regular constraints or semilinear length constraints, respectively.

For semilinear length constraints, we state the following helpful algorithmic observation.

► **Lemma 3.** *For a semilinear set S and an $n \in \mathbb{N}$, we can compute in time $O(n \text{size}(S))$ a data structure that, for every $x \in [n]$, allows us to answer whether $x \in S$ in constant time.*

A similar result can be stated for regular constraints.

► **Lemma 4.** *For a regular language $L \subseteq \Sigma^*$, given by a DFA A , accepting L , and a word $w \in \Sigma^*$, of length n , we can compute in time $O(n^2 \log \log n + \text{size}(A))$ a data structure that, for every $i, j \in [n]$, allows us to answer whether $w[i..j] \in L$ in constant time.*

► **Remark 5.** For every instance (p, w, \mathcal{C}) of $\text{MATCH}_{\text{SLS}}$, we assume that $\text{size}(C) \leq |w|$ for every $C \in \mathcal{C}$. This is justified, since without changing the solvability of the $\text{MATCH}_{\text{SLS}}$ instance, any semilinear constraint defined by some semilinear set S can be replaced by a semilinear constraint defined by the semilinear set $S \cap \{0, 1, 2, \dots, |w|\}$, which is represented by at most $|w|$ integers.

Moreover, we assume $\text{size}(C) \leq |w|^2$ for every regular constraint C for similar reasons. More precisely, a regular constraint defined by a DFA M can be replaced by a constraint defined by a DFA M' that accepts $\{w' \in L(M) \mid w' \text{ is a factor of } w\}$. The number of states and, in fact, the size of such a DFA M' can be upper bounded by $|w|^2$. For instance, the DFA M' can be the trie of all suffixes of w (constructed as in [19]), with the final states used to indicate which factors of w are valid w.r.t. C .

We emphasise that working under these assumptions allows us to focus on the actual computation done to match constrained subsequences rather than on how to deal with over-sized constraints.

4 Complexity of the Matching Problem: Initial Results

As parameters of the problem MATCH, we consider the length $|p|$ of the pattern p to be embedded as a subsequence, the number $|\mathcal{C}|$ of gap constraints, the gap description size $\text{gapsize}(\mathcal{C}) = \max\{\text{size}(C) \mid C \in \mathcal{C}\}$, and the alphabet size $|\Sigma|$. Recall that, by assumption, \mathcal{C} is always non-empty, which means that neither $|\mathcal{C}|$ nor $\text{gapsize}(\mathcal{C})$ can be zero.

For any MATCH-instance, we have that $|\mathcal{C}| \leq |p|^2$. Consequently, if $|p|$ is constant or considered a parameter, so is $|\mathcal{C}|$. This means that an upper bound with respect to parameter $|\mathcal{C}|$ also covers the upper bound with respect to parameter $|p|$, and a lower bound with respect to parameter $|p|$ also covers the lower bound with respect to parameter $|\mathcal{C}|$. Consequently, it is always enough to just consider at most one of these parameters.

For all possible parameter combinations, we can answer the respective complexity for MATCH_{REG} and MATCH_{SLS} (both when the considered parameters are bounded by a constant, or treated as parameters in terms of parameterised complexity).

From straightforward brute-force algorithms, we can conclude the following upper bounds.

► **Theorem 6.** *MATCH_{REG} and MATCH_{SLS} can be solved in polynomial time for constant $|\mathcal{C}|$. Moreover, MATCH_{REG} is fixed parameter tractable for the combined parameter $(|p|, \text{gapsize}(\mathcal{C}))$.*

These upper bounds raise the question whether MATCH_{REG} and MATCH_{SLS} are fixed parameter tractable for the single parameter $|p|$, or whether MATCH_{SLS} is at least also fixed parameter tractable for the combined parameter $(|p|, \text{gapsize}(\mathcal{C}))$, as in the case of MATCH_{REG}. Both these questions can be answered in the negative by a reduction from the CLIQUE problem.

For the k -CLIQUE problem, we get an undirected graph $G = (V, E)$ and a number $k \in [|V|]$, and we want to decide whether there is a clique of size at least k , i. e., a set $K \subseteq V$ with $|K| \geq k$ and, for every $u, v \in K$ with $u \neq v$, we have that $\{u, v\} \in E$. It is a well-known fact that k -CLIQUE is W[1]-hard. We will now sketch a parameterised reduction from k -CLIQUE to $|p|$ -MATCH_{SLS} and to $|p|$ -MATCH_{REG}.

Let $G = (V, E)$ with $|V| = n$ be a graph represented by its adjacency matrix $A = (a_{i,j})_{1 \leq i,j \leq n}$ (we assume that $a_{i,i} = 1$ for every $i \in [n]$), and let $k \in [|V|]$. It can be easily seen that G has a k -clique, if the $k \times k$ matrix containing only 1's is a principal submatrix of A , i. e., a submatrix where the set of remaining rows is the same as the set of remaining columns. This can be described as embedding $p = 01^{k^2}0$ as a subsequence into $w = 0a_{1,1}a_{1,2} \cdots a_{1,n}a_{2,1} \cdots a_{2,n} \cdots a_{n,1} \cdots a_{n,n}0$. However, the corresponding embedding e must be such that the complete i^{th} (1^k)-block of p is embedded in the same $a_{s_i,1}a_{s_i,2} \cdots a_{s_i,n}$ block of w , for some s_i . Furthermore, for every $i \in [k]$, the first 1 of the i^{th} (1^k)-block must be mapped to the $(s_1)^{\text{th}}$ 1 of $a_{s_1,1}a_{s_1,2} \cdots a_{s_1,n}$, the second 1 of the i^{th} (1^k)-block must be mapped on the $(s_2)^{\text{th}}$ 1 of $a_{s_1,1}a_{s_1,2} \cdots a_{s_1,n}$, and so on. In other words, $1 \leq s_1 < s_2 < \dots < s_k \leq n$ are the rows and columns where we map the “all-1”-submatrix; thus, $\{v_{s_1}, v_{s_2}, \dots, v_{s_k}\}$ is the clique. Obviously, we have to use the semilinear length constraints to achieve this synchronicity.

We first force $e(1) = 1$ and $e(k^2 + 2) = n^2 + 2$ by constraint $(1, k^2 + 2, L(n^2))$. In the following, we use $(i, j)_k$ and $(i, j)_n$ to refer to the position of the entries in the i -th row and j -th column of the flattened matrices in p or w respectively (e. g. $w[(i, j)_n] = a_{ij}$). In order to force that $e((i, i)_k) = (s_i, s_i)_n$ for every $i \in [k]$ and some $s_i \in [n]$, we use constraints $(1, (i, i)_k, L(0; n + 1))$, $i \in [k]$ (i. e., the first 0 of p is mapped to the first 0 of w , and then we allow only multiples of $n + 1$ between the images of $(i, i)_k$ and $(i + 1, i + 1)_k$). Next, we establish the synchronicity between the columns by requiring that the gap between $e((i, j)_k)$

and $e((i+1, j)_k)$ has a size that is one smaller than a multiple of n , which is done by constraints $((i, j)_k, (i+1, j)_k, L(n-1; n))$, $i \in [k-1]$, $j \in [k]$. Finally, the constraints $((i, 1)_k, (i, k)_k, \{0\} \cup [n-1])$, $i \in [k]$, force all $e((i, 1)_k), e((i, 2)_k), \dots, e((i, k)_k)$ into the same block $a_{s_i,1} a_{s_i,2} \dots a_{s_i,n}$. Note that in order to show the last step, we also have to argue with the previously defined constraints for synchronising the columns (more precisely, we have to show that the $e((i, 1)_k), \dots, e((i, k)_k)$ cannot overlap from one row in the next one).

This is a valid reduction from k -CLIQUE to $|p|$ -MATCH_{SLS} (note that $|p| = k^2 + 2$). We can strengthen the reduction in such a way that all constraints have even constant size (note that the constraints $((i, 1)_k, (i, k)_k, \{0\} \cup [n-1])$ are the only non-constant sized constraints, since $\{0\} \cup [n-1]$ is a semilinear set of size n). To this end, we observe that for fixed s_1 and s_k , all gap sizes $e((i, k)_k) - e((i, 1)_k)$ are the same, independent from i , namely $s_k - s_1$. Thus, we can turn the reduction into a Turing reduction by guessing this value $d = s_k - s_1$ and then replace each *non-constant* $((i, 1)_k, (i, k)_k, \{0\} \cup [n-1])$ by $((i, 1)_k, (i, k)_k, L(d))$.

In order to obtain a reduction to MATCH_{REG}, we can simply represent all semilinear constraints as regular constraints. Obviously, the corresponding DFAs are not of constant size anymore. In summary, this yields the following result.

► **Theorem 7.** *MATCH_{SLS} parameterised by $|p|$ is W[1]-hard, even for constant $\text{gapsize}(\mathcal{C})$ and binary alphabet Σ , and MATCH_{REG} parameterised by $|p|$ is W[1]-hard, even for binary alphabet Σ .*

The lower bound for MATCH_{REG} is weaker, since the parameter $\text{gapsize}(\mathcal{C})$ is not constant in the reduction. At least for a reduction from k -CLIQUE, this is to be expected, due to the fact that MATCH_{REG} is fixed parameter tractable with respect to the combined parameter $(|p|, \text{gapsize}(\mathcal{C}))$. So this leaves one relevant question open: Can MATCH_{REG} be solved in polynomial time, if the parameter $\text{gapsize}(\mathcal{C})$ is bounded by a constant? We can answer this in the negative by a reduction from a variant of the SAT-problem, which we shall sketch next.

For the problem 1-in-3-SAT, we get a set $A = \{x_1, x_2, \dots, x_n\}$ of variables and clauses $c_1, c_2, \dots, c_m \subseteq A$ with $|c_i| = 3$ for every $1 \leq i \leq m$. The task is to decide whether there is a subset $B \subseteq A$ such that $|c_i \cap B| = 1$ for every $1 \leq i \leq m$. For the sake of concreteness, we also set $c_j = \{x_{\ell_j,1}, x_{\ell_j,2}, x_{\ell_j,3}\}$ for every $j \in [m]$, and with $x_{\ell_j,1} < x_{\ell_j,2} < x_{\ell_j,3}$ for some order “ $<$ ” on A .

We transform such an 1-in-3-SAT-instance into two strings $u_A = (\mathbf{b}\#)^n(\mathbf{b}\#)^m$ and $v_A = (\mathbf{b}^2\#)^n(\mathbf{b}^3\#)^m$. For every $i \in [n]$, the i^{th} \mathbf{b} -factor of u_A and the i^{th} \mathbf{b}^2 -factor of v_A are called x_i -blocks. Analogously, we denote the last m \mathbf{b} -factors of u_A and the last m \mathbf{b}^3 -factors of v_A as c_j -blocks for $j \in [m]$.

Obviously, if $u_A \preceq_e v_A$ for some embedding $e: [|u_A|] \rightarrow [|v_A|]$, then the single \mathbf{b} of u_A 's x_i -block is mapped to either the first or the second \mathbf{b} of v_A 's x_i -block, and the single \mathbf{b} of u_A 's c_j -block is mapped to either the first or the second or the third \mathbf{b} of v_A 's c_j -block. Thus, the embedding e can be interpreted as selecting a set $B \subseteq A$ (where mapping u_A 's x_i -block to the *second* \mathbf{b} of v_A 's x_i -block is interpreted as $x_i \in B$), and selecting either the first or second or third element of c_j (depending on whether u_A 's c_j -block is mapped to the first, second or third \mathbf{b} of v_A 's c_j -block). We can now introduce a set of regular gap constraints that enforce the necessary synchronicity between B and the elements picked from the clauses: Assume that x_i is the p^{th} element of c_j . If e maps u_A 's x_i -block to the second \mathbf{b} of v_A 's x_i -block, then e must map u_A 's c_j -block to the p^{th} \mathbf{b} of v_A 's c_j -block, and if e maps u_A 's x_i -block to the first \mathbf{b} of v_A 's x_i -block, then e must map u_A 's c_j -block to the q^{th} \mathbf{b} of v_A 's c_j -block for some $q \in \{1, 2, 3\} \setminus \{p\}$. For example, if $x_{\ell_j,2} = x_i$ for some $i \in [n]$ and $j \in [m]$, i.e., the second element of c_j is x_i , then we add a regular gap

constraint $(i', j', L_{i', j'})$, where i' and j' are the positions of u_A 's x_i -block and u_A 's c_j -block, and $L_{i', j'} = \{\mathbf{b}w\#, \#w\#\mathbf{b}, \mathbf{b}w\#\mathbf{b}\mathbf{b} \mid w \in \{\mathbf{b}, \#\}^*\}$. If e maps u_A 's x_i -block to the second \mathbf{b} of v_A 's x_i -block, then the gap between positions i' and j' must start with $\#$; thus, due to the gap constraint, it must be of the form $\#w\#\mathbf{b}$, which is only possible if e maps u_A 's c_j -block to the second \mathbf{b} of v_A 's c_i -block. If e maps u_A 's x_i -block to the first \mathbf{b} of v_A 's x_i -block, then the gap must be of the form $\mathbf{b}w\#$ or $\mathbf{b}w\#\mathbf{b}\mathbf{b}$, which means that e must map u_A 's c_j -block to the first or third \mathbf{b} of v_A 's c_i -block. Similar gap constraints can be defined for the case that x_i is the first or third element of c_j . Hence, we can define gap constraints such that there is an embedding $e : \llbracket u_A \rrbracket \rightarrow \llbracket v_A \rrbracket$ with $u_A \preceq_e v_A$ and e satisfies all the gap constraints if and only if the 1-in-3-3SAT-instance is positive. Independent of the actual 1-in-3-3SAT-instance, the gap languages can be represented by DFAs with at most 8 states. This shows the following result.

► **Theorem 8.** *MATCH_{REG} is NP-complete, even for binary alphabet Σ and with gap-constraints that can be represented by DFAs with at most 8 states.*

5 Complexity of the Matching Problem: A Finer Analysis

Two representations of constraints. We start by defining two (strongly related) natural representations of the set of constraints which is given as input to the matching problem. Intuitively, both these representations facilitate the understanding of a set of constraints. Then, we see how these representations can be used to approach the MATCH problem.

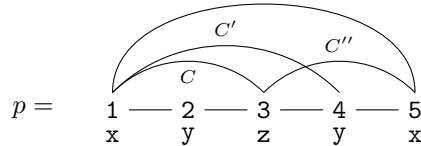
The Interval Structure of Sets of Constraints. For a constraint $C = (a, b, L)$ we define $\text{interval}(C) := [a, b - 1]$. If we have another constraint $C' = (a', b', L')$ (with $a \neq a'$ or $b \neq b'$), we say that C is contained in C' , written $C \sqsubset C'$, if $\text{interval}(C) \subsetneq \text{interval}(C')$. Because this order is derived from the inclusion order \subsetneq , it is also a strict partial order. We denote the corresponding covering relation with \sqsupseteq . Furthermore, we say that C and C' intersect, if $\text{interval}(C) \cap \text{interval}(C') \neq \emptyset$, and they are not comparable w.r.t. \sqsubset , i.e., neither of them contains the other. Importantly, because we have $b \notin \text{interval}(C)$, in the case $b = a'$, the constraints C and C' do not intersect.

The Graph Structure of Sets of Constraints. For a string p , of length m , and a set of constraints \mathcal{C} on p , we define a graph $G_p = (V_p, E_p)$ as follows. The set of vertices V_p of G_p is the set of numbers $\{1, \dots, m\}$, corresponding to the positions of p . We define the set of undirected edges E_p as follows: $E_p = \{(a, b) \mid (a, b, L) \in \mathcal{C}\} \cup \{(i, i+1) \mid i \in [m-1]\} \cup \{(1, m)\}$. Note that in the case when we have a constraint $C = (i, i+1, L)$, for some $i \in [m-1]$ and L (respectively, a constraint $C(1, m, L)$) we will still have a single edge connecting the nodes i and $i+1$ (respectively, 1 and m) as E_p is constructed by a set union of two sets, so the elements in the intersection of the two sets are kept only once in the result. Moreover, we can define a labelling function on the edges of G by the following three rules: $\text{label}((i, j)) = C$, if there exists $C \in \mathcal{C}$ with $C = (i, j, L)$; $\text{label}((i, i+1)) = (i, i+1, \Sigma^*)$, for $i \in [m-1]$, if there exists no $C \in \mathcal{C}$ such that $C = (i, i+1, L)$; and $\text{label}((1, m)) = (1, m, \{w \in \Sigma^* \mid |w| \geq m-2\})$, if there exists no $C \in \mathcal{C}$ such that $C = (1, m, L)$. Clearly, all respective labels can be expressed trivially both as regular languages or as semilinear sets. Intuitively, the edges of G which correspond to constraints of \mathcal{C} are labelled with the respective constraints. The other edges have trivial labels: the label of the edges of the form $(i, i+1)$ express that, in an embedding of p in the string v (as required in MATCH), the embedding of position $i+1$ is

to the right of the embedding of position i ; the label of edge $(1, m)$ simply states that at least $m - 2$ symbols should occur between the embedding of position 1 and the embedding of position m , therefore allowing for the entire pattern to be embedded.

Further, it is easy to note that this graph admits a Hamiltonian cycle, which traverses the vertices $1, 2, \dots, m$ in this order. In the following, we also define two-dimensional drawing of the graph G_p as a *half-plane arc diagram*: the vertices $1, 2, \dots, m$ are represented as points on a horizontal line ℓ , with the edges $(i, i + 1)$, for $i \leq m - 1$ being segments of unit-length on that line, spanning between the respective vertices i and $i + 1$; all other edges (i, j) are drawn as semi-circles, whose diameter is equal to the length of $(j - i)$, drawn in the upper half-plane with respect to the line ℓ . In the following, we will simply call this diagram arc-diagram, without explicitly recalling that all semicircles are drawn in the same half-plane with respect to ℓ . In this diagram associated to G_p , we say that two edges cross if and only if they intersect in a point of the plane which is not a vertex of G_p . See also [18] and the references therein for a discussion on this representation of graphs.

In Figure 1 we see an example: We have $p = \mathbf{xyzyx}$ and $\mathcal{C} = \{C = (1, 3, L_1), C' = (1, 4, L_1), C'' = (3, 5, L_2)\}$. Thus, $|p| = 5$ and G_p has the vertices $\{1, \dots, 5\}$ and the edges $(1, 3), (1, 4), (3, 5)$ corresponding to the constraints of \mathcal{C} , as well as the edges $(1, 2), (2, 3), (3, 4), (4, 5), (1, 5)$ (note that the edges are undirected, and the trivial labels are omitted for the sake of readability). The figure depicts the arc diagram associated to this graph (with the semi-circles flattened a bit, for space reasons). In the interval representation of these constraints, we can see that C is contained in C' (as $\text{interval}(C) = [1, 2] \subsetneq \text{interval}(C') = [1, 3]$). Furthermore, C' and C'' intersect ($[1, 3] \cap [3, 4] \neq \emptyset$), while C and C'' do not ($[1, 2] \cap [3, 4] = \emptyset$). Note that two constraints (such as C and C'' in our example) might not intersect (according to the interval representation), although the edges that correspond to them in the graph representation share a common vertex; in particular, two constraints intersect if and only if the corresponding edges cross.



■ **Figure 1** Relations between constraints.

The two representations defined above are clearly very strongly related. However, the graph-representation allows us to define a natural structural parameter for subsequences with constraints, while the interval-representation will allow us to introduce a class of subsequences with constraints which can be matched efficiently.

In the following, by $\text{MATCH}_{\mathcal{L}, \mathcal{G}}$ we denote the problem MATCH , where all gap constraints are from the class of languages \mathcal{L} , with $\mathcal{L} \in \{\text{REG}, \text{SLS}\}$, and the graphs corresponding to the input gap constraints are all from the class \mathcal{G} .

Vertex separation number and its relation to Match. Given a linear ordering $\sigma = (v_1, \dots, v_m)$ of the vertices of a graph G with m vertices, the vertex separation number of σ is the smallest number s such that, for each vertex v_i (with $i \in [m]$), at most s vertices of v_1, \dots, v_{i-1} have some v_j , with $j \geq i$, as neighbour. The vertex separation number $\text{vsn}(G)$ of G is the minimum vertex separation number over all linear orderings of G . The vertex separation number was introduced in [20] (see also [21] and the references therein) and was shown (e.g., in [9]) to be equal to the pathwidth of G .

Let us briefly overview the problem of computing the vertex separation number of graphs. Firstly, we note that checking, given a graph G with n vertices and a number k , whether $\text{vsn}(G) \leq k$ is NP-complete, as it is equivalent to checking whether the pathwidth of G is at most k . We can show that this problem remains intractable even when we restrict it to the class of graphs with a Hamiltonian cycle, and this cycle is given as input as well.

However, this problem is linear fixed parameter tractable w.r.t. the parameter k : deciding, for a given graph G with n vertices and a constant number k , if $\text{vsn}(G) \leq k$ and, if so, computing a linear ordering of the vertices with vertex separation number at most k can be solved in $O(n)$ time, where the constant hidden by the O -notation depends superexponentially on k . This follows from the results of [9], where the relation between pathwidth and vertex separation number is established, and [8], where it is shown that, for constant k , one can check in linear time $O(n)$ if a graph with n vertices has pathwidth at most k , and, if so, produce a path decomposition of that graph of width at most k .

For a constant k , let \mathcal{V}_k be the class of all graphs G with $\text{vsn}(G) \leq k$. We can show the following meta-theorem.

► **Theorem 9.** *Let $k \geq 1$ be a constant integer and let $\mathcal{L} \in \{\text{SLS}, \text{REG}\}$. Then, $\text{MATCH}_{\mathcal{L}, \mathcal{V}_k}$ can be solved in polynomial time: $O(m^2 n^{k+1})$, in the case of SLS-constraints, and $O(m^2 n^{k+1} + m^2 n^2 \log \log n)$, in the case of REG-constraints. Moreover, $\text{MATCH}_{\mathcal{L}, \mathcal{V}_k}$ parameterised by k is W[1]-hard.*

Due to space restrictions, we only sketch the proof. The matching algorithm implements a dynamic programming approach. We choose an ordering of the vertices of the graph representing \mathcal{C} , with vsn at most k . These vertices are, in fact, positions of p , so we find, for q from 1 to m , embeddings for the first q positions of this ordering in w , such that all the constraints involving only these positions are fulfilled. Given that the vsn of the respective ordering is bounded by k , we can compute efficiently the embeddings of the first q positions by extending the embeddings for the first $q - 1$ positions, as, when considering a new position of our ordering, and checking where it can be embedded, only k of the previously embedded positions are relevant. As such, the embeddings of the first $q - 1$ positions of the ordering, which are relevant for computing the embeddings of its first q positions, can be represented using an $O(n^k)$ size data-structure, and processed in $O(n^k \text{poly}(n, m))$ time. This leads to a polynomial time algorithm, with a precise runtime as stated above. The lower bound follows from the reduction showing Theorem 7.

Non-intersecting constraints and Match. We have shown that MATCH can be solved efficiently if the input gap constraints are represented by graphs with bounded vsn . However, while this condition is sufficient to ensure that MATCH can be solved efficiently (as long as the constraint-languages are in P), it is not necessary. We will exhibit in the following a class \mathcal{H} of gap constraints, which contains graphs with arbitrarily large vsn , and for which $\text{MATCH}_{\mathcal{L}, \mathcal{H}}$ can be solved in polynomial time, for $\mathcal{L} \in \{\text{REG}, \text{SLS}\}$.

More precisely, in the following, we will consider the class \mathcal{H} of non-intersecting gap constraints. That is, we consider MATCH where the input consists of two strings $v \in \Sigma^*$ and $p \in \Sigma^*$ and a non-empty set \mathcal{C} of gap constraints, where for any $C, C' \in \mathcal{C}$ we have that $\text{interval}(C) \cap \text{interval}(C') \in \{\text{interval}(C), \text{interval}(C'), \emptyset\}$. It is immediate that \mathcal{H} , the class of non-intersecting gap constraints, can be described as the class of gap constraints which are represented by outerplanar graphs which have a Hamiltonian cycle: the arc diagram constructed for these gap constraints is already outerplanar: if $C = (a, b, L)$ and $C' = (a', b', L')$ are two non-intersecting constraints of some set of constraints \mathcal{C} , then, in the graph representation of this set of constraints based on arc diagrams, the edges (a, b) and (a', b') do not cross (although they might share a common vertex).

Moreover, an outerplanar graph which admits a Hamiltonian cycle can be represented canonically as an arc diagram of a set of non-intersecting gap constraints. It is a folklore result that if an outerplanar graph has a Hamiltonian cycle then the outer face forms its unique Hamiltonian cycle. Moreover, every drawing of a graph in the plane may be deformed into an arc diagram without changing its number of crossings [45], and, in the case of outerplanar graphs this means the following. For an outerplanar graph G , we start with a drawing of G witnessing its outerplanarity. Assume that, after a potential renaming, there exists a traversal of the Hamiltonian cycle of the outerplanar graph (i.e., of its outer face) which consists of the vertices $1, 2, \dots, n$, in this order. We simply reposition these vertices, in the same order $1, 2, \dots, n$, on a horizontal line ℓ , such that consecutive vertices on the line are connected by edges of length 1, and then the edge $(1, n)$ is deformed so that it becomes a semicircle of diameter $n - 1$ connecting the respective vertices, in the upper half-plane w.r.t. ℓ . Further, each edge (a, b) is deformed to become a semicircle above the line of the vertices, whose diameter equals the distance between vertex a and vertex b . By the result of [45], the edges of this graph do not cross in this representation, as the initial graph was outerplanar. But the resulting diagram is, clearly, the arc diagram corresponding to a set of non-intersecting gap constraints.

Based on the above, we can make a series of observations. Firstly, as one can recognize outerplanar graphs in linear time [56], we can also decide in linear time whether a set of constraints is non-crossing. Secondly, according to [16], there are outerplanar graphs with arbitrarily large pathwidth, so with arbitrarily large vsn. This means that there are sets of non-intersecting gap constraints whose corresponding graph representations have arbitrarily large vsn. Thirdly, the number of constraints in a set of non-intersecting gap constraints is linear in the length of the string constrained by that set (as the number of edges in an outerplanar graph with n vertices is at most $2n - 3$).

We can show the following theorem (here we just sketch the proof, and only in the case of SLS-constraints, as the REG-constraints case is identical).

► **Theorem 10.** *$\text{MATCH}_{\text{SLS}, \mathcal{H}}$ can be solved in time $O(n^\omega K)$ and $\text{MATCH}_{\text{REG}, \mathcal{H}}$ can be solved in time $O(n^\omega K + n^2 K \log \log n)$, where K is the number of constraints in the input set of constraints \mathcal{C} and $O(n^\omega)$ is the time needed to multiply two boolean matrices of size $n \times n$.*

As said, we sketch the algorithm solving $\text{MATCH}_{\text{SLS}, \mathcal{H}}$ in the stated complexity. Assume that the input words are $w \in \Sigma^n$ and $p \in \Sigma^m$, and $\mathcal{C} = (C_1, \dots, C_K)$, with $C_i = (a_i, b_i, L_i)$ for $i \in [K]$. Firstly, we add a constraint $C = (1, m, L(m - 2, 1))$ to \mathcal{C} if it does not contain any constraint having the first two components $(1, m)$. So, in the following, we will assume w.l.o.g. that such a constraint $(1, m, \cdot)$ always exists in \mathcal{C} . Moreover, the number of constraints in \mathcal{C} is $O(m)$ (as the graph representation of \mathcal{C} is outerplanar).

As a first phase in our algorithm, we build the data structures from Lemma 3. Hence, by Remark 5, after an $O(n^2 K)$ -time preprocessing we can answer queries “is $w[i..j] \in L_k$?” in $O(1)$ time, for all $i, j \in [n], k \in [K]$.

After this, the algorithm proceeds as follows. Because the set of constraints \mathcal{C} is non-intersecting, one can build in linear time the Hasse-diagram of the set of intervals associated with the set of constraints \mathcal{C} (w.r.t. the interval-inclusion relation), and this diagram is a tree, whose root corresponds to the single constraint of the form $(1, m, \cdot)$. Further, the algorithm uses a dynamic programming strategy to find matches for the constraints of \mathcal{C} in a bottom-up fashion with respect to the Hasse-diagram of this set. The algorithm maintains the matches for each constraint $C = (a, b, L)$ as a Boolean $n \times n$ matrix, where the element on position (i, j) of that matrix is true if and only if there exists a way to embed $p[a..b]$ in $w[i..j]$, such that $p[a]$ is mapped to $w[i]$ and $p[b]$ to $w[j]$ in the respective embedding, and this embedding

also fulfils C and all the constraints occurring in the sub-tree of root C in the Hasse-diagram. This matrix can be computed efficiently, by multiplying the matrices corresponding to the children of C (and a series of matrices corresponding to the unconstrained parts of $p[a..b]$). As the number of nodes in this tree is $O(K)$, the whole process of computing the respective matrices for all nodes of the tree requires $O(K)$ matrix multiplications, i.e., $O(n^\omega K)$ time in total. Finally, one needs to see if there is a match of $p[1..m]$ to some factor of v , which can be checked in $O(n^2)$ by simply searching in the matrix computed for the root of the diagram.

The following lower bound is shown by a reduction from 3-OV.

► **Theorem 11.** *For $\mathcal{L} \in \{\text{REG}, \text{SLS}\}$, then $\text{MATCH}_{\mathcal{L}, \mathcal{H}}$, cannot be solved in $\mathcal{O}(|w|^g |\mathcal{C}|^h)$ time with $g + h < 3$, unless *SETH* fails.*

The reduction proving this hardness result works as follows. In 3-OV, we are given three sets $A = \{\vec{a}_1, \dots, \vec{a}_n\}$, $B = \{\vec{b}_1, \dots, \vec{b}_n\}$ and $C = \{\vec{c}_1, \dots, \vec{c}_n\}$ with elements from $\{0, 1\}^d$, and want to determine whether there exist $i^*, j^*, k^* \in [n]$, such that $\sum_{\ell=1}^j \vec{a}_{i^*}[\ell] \cdot \vec{b}_{j^*}[\ell] \cdot \vec{c}_{k^*}[\ell] = 0$. This is achieved by encoding our input sets over a constant size alphabet, via two functions C_p and C_w , into a pattern p and a text w , respectively, as well as a set of constraints \mathcal{C} , such that the answer to the 3-OV problem is positive if and only if p is a \mathcal{C} -subsequence of w . Basically, the encoding of each d -dimensional vector of A (respectively, B and C) is done via C_p (respectively, C_w), in such a way that (when no constraints are considered) $C_p(\vec{v})$ is a subsequence of $C_w(\vec{v}')$ for any $\vec{v}, \vec{v}' \in \{0, 1\}^d$. Further, \bar{C}_p and \bar{C}_w are mirrored versions of these encodings (both for bits and vectors), where the order of the characters in the output is inverted. We can then use the original encoding for one part of the pattern and the text and the mirrored encoding for the other part. Then, we encode the set A in $p := \bar{C}_p(\vec{a}_n) \dots \bar{C}_p(\vec{a}_1) \# C_p(\vec{a}_1) \dots C_p(\vec{a}_n)$ and the sets B and C in $w := \bar{w}_0 \# \bar{C}_w(\vec{c}_n) \dots \bar{C}_w(\vec{c}_1) \# \bar{w}_0 \# w_0 \# C_w(\vec{b}_1) \dots C_w(\vec{b}_n) \# w_0$ (where \bar{x} denotes the mirror image of x , and w_0 is a suitably chosen padding). To finalise our construction, we can define constraints which ensure that an embedding of p in w is possible if and only if there exist some i^*, j^*, k^* such that $C_p(a_{i^*})$ is embedded in $C_w(b_{j^*})$, and $\bar{C}_p(a_{i^*})$ in $\bar{C}_p(c_{k^*})$, while all the other strings $\bar{C}_p(a_t)$ are embedded in the paddings. Moreover, additional constraints ensure that the simultaneous embedding of $C_p(a_{i^*})$ in $C_w(b_{j^*})$ and of $\bar{C}_p(a_{i^*})$ in $\bar{C}_p(c_{k^*})$ is only possible if and only if for each component $u \in [d]$ with $a_{i^*}[u] \neq 0$, we have that $b_{j^*}[u] = 0$ or $c_{k^*}[u] = 0$.

We conclude by noting that, while this is not a tight lower bound with respect to the upper bound shown in Theorem 10, finding a polynomially stronger lower bound (i.e., replacing in the statement of Theorem 11 the condition $g + h < 3$ with $g + h < 3 + \delta$, for some $\delta > 0$) would show that matrix multiplication in quadratic time is not possible, which in turn would solve a well-researched open problem. Indeed, the algorithm from Theorem 10 consists in a reduction from $\text{MATCH}_{\mathcal{L}, \mathcal{H}}$ to $\mathcal{O}(|C|)$ instances of matrix multiplication, for quadratic matrices of size $|w|$, so a better lower bound would mean that at least one of these multiplications must take more than quadratic time.

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 Duncan Adamson, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Longest common subsequence with gap constraints. In *Combinatorics on Words - 14th International Conference, WORDS 2023, Umeå, Sweden, June 12-16, 2023, Proceedings*, pages 60–76, 2023. doi:10.1007/978-3-031-33180-0_5.
- 4 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.
- 5 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.
- 6 Ricardo A. Baeza-Yates. Searching subsequences. *Theor. Comput. Sci.*, 78(2):363–376, 1991.
- 7 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.
- 8 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 9 Hans L. Bodlaender. A partial k -arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.
- 10 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.
- 11 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proc. SODA 2018*, pages 1216–1235, 2018.
- 12 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.
- 13 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.
- 14 Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranajit B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.
- 15 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: <http://mitpress.mit.edu/books/introduction-algorithms>.
- 16 David Coudert, Florian Huc, and Jean-Sébastien Sereni. Pathwidth of outerplanar graphs. *J. Graph Theory*, 55(1):27–41, 2007. doi:10.1002/JGT.20218.
- 17 Joel D. Day, Maria Kosche, Florin Manea, and Markus L. Schmid. Subsequences with gap constraints: Complexity bounds for matching and analysis problems. In *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, pages 64:1–64:18, 2022. doi:10.4230/LIPICs.ISAAC.2022.64.
- 18 Hristo N. Djidjev and Imrich Vrto. Crossing numbers and cutwidths. *J. Graph Algorithms Appl.*, 7(3):245–251, 2003. doi:10.7155/JGAA.00069.
- 19 Shiri Dori and Gad M. Landau. Construction of aho corasick automaton in linear time for integer alphabets. *Inf. Process. Lett.*, 98(2):66–72, 2006. doi:10.1016/J.IPL.2005.11.019.
- 20 John A. Ellis, Ivan Hal Sudborough, and Jonathan S. Turner. Graph separation and search number. In *Proc. 1983 Allerton Conf. on Communication, Control, and Computing*, 1983.
- 21 John A. Ellis, Ivan Hal Sudborough, and Jonathan S. Turner. The vertex separation and search number of a graph. *Inf. Comput.*, 113(1):50–79, 1994. doi:10.1006/INCO.1994.1064.

- 22 Pamela Fleischmann, Sungmin Kim, Tore Koß, Florin Manea, Dirk Nowotka, Stefan Siemer, and Max Wiedenhöft. Matching patterns with variables under simon’s congruence. In *Reachability Problems - 17th International Conference, RP 2023, Nice, France, October 11-13, 2023, Proceedings*, pages 155–170, 2023. doi:10.1007/978-3-031-45286-4_12.
- 23 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.
- 24 André Frochoux and Sarah Kleest-Meißner. Puzzling over subsequence-query extensions: Disjunction and generalised gaps. In *Proceedings of the 15th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW 2023), Santiago de Chile, Chile, May 22-26, 2023*, 2023. URL: <https://ceur-ws.org/Vol-3409/paper3.pdf>.
- 25 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.
- 26 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.
- 27 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.
- 28 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.
- 29 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.
- 30 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.
- 31 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.
- 32 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 33 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, 29th March-1st April, 2022 Edinburgh, UK*, 2022.
- 34 Sarah Kleest-Meißner, Rebecca Sattler, Markus L. Schmid, Nicole Schweikardt, and Matthias Weidlich. Discovering multi-dimensional subsequence queries from traces - from theory to practice. In *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS), 06.-10. März 2023, Dresden, Germany, Proceedings*, pages 511–533, 2023. doi:10.18420/BTW2023-24.
- 35 Maria Kosche, Tore Koß, Florin Manea, and Viktoriya Pak. Subsequences in bounded ranges: Matching and analysis problems. In Anthony W. Lin, Georg Zetsche, and Igor Potapov, editors, *Reachability Problems - 16th International Conference, RP 2022, Kaiserslautern, Germany, October 17-21, 2022, Proceedings*, volume 13608 of *Lecture Notes in Computer Science*, pages 140–159. Springer, 2022. doi:10.1007/978-3-031-19135-0_10.

- 36 Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Combinatorial algorithms for subsequence matching: A survey. In Henning Bordihn, Géza Horváth, and György Vaszil, editors, *Proceedings 12th International Workshop on Non-Classical Models of Automata and Applications, NCMA 2022, Debrecen, Hungary, August 26-27, 2022*, volume 367 of *EPTCS*, pages 11–27, 2022. doi:10.4204/EPTCS.367.2.
- 37 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.
- 38 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.
- 39 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.
- 40 Julien Leroy, Michel Rigo, and Manon Stipulanti. Generalized Pascal triangle for binomial coefficients of words. *Electron. J. Combin.*, 24(1.44):36 pp., 2017.
- 41 Chun Li and Jianyong Wang. Efficiently mining closed subsequences with gap constraints. In *SDM*, pages 313–322. SIAM, 2008.
- 42 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.
- 43 David Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, April 1978.
- 44 Alexandru Mateescu, Arto Salomaa, and Sheng Yu. Subword histories and Parikh matrices. *J. Comput. Syst. Sci.*, 68(1):1–21, 2004.
- 45 T.A.J. Nicholson. Permutation procedure for minimising the number of crossings in a network. *Proceedings of the Institution of Electrical Engineers*, 115:21–26(5), January 1968.
- 46 Rohit J Parikh. Language generating devices. *Quarterly Progress Report*, 60:199–212, 1961.
- 47 M. Praveen, Philippe Schnoebelen, Julien Veron, and Isa Vialard. On the piecewise complexity of words and periodic words. In *SOFSEM 2024: Theory and Practice of Computer Science - 48th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2024, Cochem, Germany, February 19-23, 2024, Proceedings*, pages 456–470, 2024. doi:10.1007/978-3-031-52113-3_32.
- 48 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.
- 49 Michel Rigo and Pavel Salimov. Another generalization of abelian equivalence: Binomial complexity of infinite words. *Theor. Comput. Sci.*, 601:47–57, 2015.
- 50 Arto Salomaa. Connections between subwords and certain matrix mappings. *Theoret. Comput. Sci.*, 340(2):188–203, 2005.
- 51 Philippe Schnoebelen and Julien Veron. On arch factorization and subword universality for words and compressed words. In *Combinatorics on Words - 14th International Conference, WORDS 2023, Umeå, Sweden, June 12-16, 2023, Proceedings*, pages 274–287, 2023. doi:10.1007/978-3-031-33180-0_21.
- 52 Shinnosuke Seki. Absoluteness of subword inequality is undecidable. *Theor. Comput. Sci.*, 418:116–120, 2012. doi:10.1016/J.TCS.2011.10.017.
- 53 Alan C. Shaw. Software descriptions with flow expressions. *IEEE Trans. Software Eng.*, 4(3):242–254, 1978. doi:10.1109/TSE.1978.231501.
- 54 Imre Simon. *Hierarchies of events with dot-depth one* — *Ph.D. thesis*. University of Waterloo, 1972.
- 55 Imre Simon. Piecewise testable events. In *Autom. Theor. Form. Lang., 2nd GI Conf.*, volume 33 of *LNCS*, pages 214–222, 1975.
- 56 Manfred Wieggers. Recognizing outerplanar graphs in linear time. In Gottfried Tinhofer and Gunther Schmidt, editors, *Graphtheoretic Concepts in Computer Science, International Workshop, WG '86, Bernried, Germany, June 17-19, 1986, Proceedings*, volume 246 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 1986. doi:10.1007/3-540-17218-1_57.

- 57 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.
- 58 Virginia Vassilevska Williams. *On some fine-grained questions in algorithms and complexity*, pages 3447–3487. World Scientific, 2018. doi:10.1142/9789813272880_0188.
- 59 Georg Zetsche. The complexity of downward closure comparisons. In *Proc. ICALP 2016*, volume 55 of *LIPICs*, pages 123:1–123:14, 2016.
- 60 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.