Discovering Event Queries from Traces: Laying Foundations for Subsequence-Queries with Wildcards and Gap-Size Constraints

Sarah Kleest-Meißner
Humboldt-Universität zu Berlin, Germany

Rebecca Sattler
Humboldt-Universität zu Berlin, Germany

Markus L. Schmid
Humboldt-Universität zu Berlin, Germany

Nicole Schweikardt
Humboldt-Universität zu Berlin, Germany

Matthias Weidlich
Humboldt-Universität zu Berlin, Germany

Abstract

We introduce subsequence-queries with wildcards and gap-size constraints (swg-queries, for short) as a tool for querying event traces. An swg-query q is given by a string s over an alphabet of variables and types, a global window size w, and a tuple $c = ((c_1^-, c_1^+), (c_2^-, c_2^+), \ldots, (c_{|s|-1}^-, c_{|s|-1}^+))$ of local gap-size constraints over $\mathbb{N} \times (\mathbb{N} \cup \{\infty\})$. The query q matches in a trace t (i. e., a sequence of types) if the variables can uniformly be substituted by types such that the resulting string occurs in t as a subsequence that spans an area of length at most w, and the i^{th} gap of the subsequence (i. e., the distance between the i^{th} and $(i+1)^{\text{th}}$ position of the subsequence) has length at least c_i^- and at most c_i^+. We formalise and investigate the task of discovering an swg-query that describes best the traces from a given sample $\mathcal{S}$ of traces, and we present an algorithm solving this task. As a central component, our algorithm repeatedly solves the matching problem (i. e., deciding whether a given query q matches in a given trace t), which is an NP-complete problem (in combined complexity). Hence, the matching problem is of special interest in the context of query discovery, and we therefore subject it to a detailed (parameterised) complexity analysis to identify tractable subclasses, which lead to tractable subclasses of the discovery problem as well. We complement this by a reduction proving that any query discovery algorithm also yields an algorithm for the matching problem. Hence, lower bounds on the complexity of the matching problem directly translate into according lower bounds of the query discovery problem. As a proof of concept, we also implemented a prototype of our algorithm and tested it on real-world data.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Theory of computation → Data structures and algorithms for data management; Theory of computation → Problems, reductions and completeness; Information systems → Data mining

Keywords and phrases event queries on traces, pattern queries on strings, learning descriptive queries, complexity of query evaluation and query learning

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.18

Funding *Sarah Kleest-Meißner*: Supported by the German Research Foundation (DFG), CRC 1404: "FONDA: Foundation of Workflows for Large-Scale Scientific Data Analysis".
Rebecca Sattler: Supported by the German Research Foundation (DFG), CRC 1404: "FONDA: Foundation of Workflows for Large-Scale Scientific Data Analysis".
Markus L. Schmid: Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) – project number 416776735 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 416776735).

1 Introduction

Event stream processing emerged as a computational paradigm that is based on the continuous evaluation of queries over event streams [10, 22]. Respective systems enable the definition of queries that detect patterns of events, i.e., a match of a query is a joint occurrence of events that satisfy certain properties. Common languages for the definition of event queries, as reviewed in [22, 5], include means to specify such properties related to the order of events, their types, as well as a time window for their joint occurrence.

Event stream processing has traditionally been used for reactive applications in diverse domains, reaching from infrastructure monitoring [7] through financial trading [29] to urban transportation [6]. Here, event queries are specified by expert users to detect situations of interest. For example, consider the case of monitoring a large compute cluster based on events that indicate that a particular *job* was *submitted*, *scheduled*, *suspended*, *resumed*, and *updated* while running, before it either *completes* successfully, is *evicted* due to preemption, *fails* with an error, or is *aborted* manually. In addition, events may carry information on the job's *priority* and the *machine* on which the job runs. Given such event streams, abnormal job execution materialises as an event pattern. For instance, a job may be evicted and rescheduled twice on the same machine, with low and unchanged priority, before it eventually fails. Adopting the SASE language [31], Listing 1 formalises this query.

Listing 1 Event query defined in the SASE language [31]. A match is a sequence of events of the respective transition types (`Evict`, `Schedule`, `Fail`), all being related to the same job (with `[job]` being a short-hand for `a.job=b.job`, `a.job=c.job`, etc.) and to the same machine (`[machine]`), while the priority remains unchanged (`b.prio=low`, `d.prio=low`); all within one hour.

```
PATTERN SEQ(Evict a, Schedule b, Evict c, Schedule d, Fail e)
WHERE [job] AND [machine] AND b.prio=low AND d.prio=low
WITHIN 1h
```

In pro-active applications where an event query shall anticipate a situation of interest to prepare for it accordingly [4], users often only know when a specific situation occurred, but they have no or only partial knowledge on the patterns that indicate that the situation will materialise soon. In the above example, a user may be aware that certain jobs fail execution, and potentially possesses anecdotal evidence that only jobs with low priority are affected, but it is not known that repeated eviction and re-scheduling on the same machine may serve to forecast this failure. Against this background, it was suggested to automatically infer the event patterns of interest from historic event data [21, 24], which can then be interpreted and validated by a user, thereby mitigating the risk of overfitting and enabling traceability.

Specifically, historic event data is first split into *traces* that are considered independent observations (e.g., events may be grouped by *jobs* and *machines*), before the traces related to the situation of interest are identified (e.g., based on the presence of *failure* events). See also Figure 1. Those then serve as a *sample* of positive examples for the discovery process of event queries that are (i) *correct* (they match a share of the sample that is above a given support threshold), and (ii) *descriptive* (any query that is more specific is no longer correct).

In this paper, we introduce *subsequence-queries with wildcards and gap-size constraints* (*swg-queries*) as a formal model that covers the essence of the task of event query discovery. Intuitively, an swg-query is a string over an alphabet of variables and types, a global window size and a tuple of local gap-size constraints. It matches a trace, i.e., a sequence of types, if its variables can be substituted by types, such that the resulting string represents a subsequence of the trace, and the global window size and the local gap-size constraints are satisfied.

```
Schedule(time=21, job=3, machine=m5v, prio=low)
Evict(time=42, job=3, machine=m5v)
Schedule(time=52, job=5, machine=m5v, prio=high)
Schedule(time=112, job=3, machine=m5v, prio=low)
Update(time=142, job=5, machine=m5v)
Complete(time=217, job=5, machine=m5v)
Suspend(time=276, job=3, machine=m5v)
```

$t_1 = \mathsf{SchLow}\ \ \mathsf{E}\ \ \mathsf{SchLow}\ \ \mathsf{Sus}$
$t_2 = \mathsf{SchHigh}\ \ \mathsf{U}\ \ \mathsf{C}$

(a) Events and their representation as traces, i.e., sequences of types, per job and machine.

$t_1 = \mathsf{SchLow}\ \ \mathsf{E}\ \ \mathsf{SchLow}\ \ \mathsf{Sus}$
$t_2 = \mathsf{SchHigh}\ \ \mathsf{U}\ \ \mathsf{C}$
$t_3 = \mathsf{SchHigh}\ \ \mathsf{E}\ \ \mathsf{SchHigh}\ \ \mathsf{U}\ \ \mathsf{E}\ \ \mathsf{F}$
$t_4 = \mathsf{SchHigh}\ \ \mathsf{U}\ \ \mathsf{U}\ \ \mathsf{C}$
$t_5 = \mathsf{SchLow}\ \ \mathsf{E}\ \ \mathsf{SchLow}\ \ \mathsf{Sus}\ \ \mathsf{Res}\ \ \mathsf{E}\ \ \mathsf{A}$
$t_6 = \mathsf{SchLow}\ \ \mathsf{U}\ \ \mathsf{U}\ \ \mathsf{E}\ \ \mathsf{SchLow}\ \ \mathsf{E}\ \ \mathsf{A}$

(b) Database of six traces; t_3, t_5, and t_6 denote failed (F) and aborted (A) executions.

Figure 1 Illustration of the setting of event query discovery.

Let us illustrate how the discovery of event queries defined in common languages for event stream processing translates into the discovery of swg-queries.

▶ **Example 1.1.** The seven events of Figure 1a indicate lifecycle transitions of two jobs. Encoding the events by types that model the lifecycle transitions, potentially combined with additional payload data (e.g., `Schedule` events are modelled by types `SchLow` and `SchHigh`, depending on the assigned priority), we derive two traces t_1 and t_2 (the order of types follows from the events' occurrence times). Note that we partitioned the events by the corresponding job such that t_1 and t_2 consists only of events associated to job 3 and 5, respectively. Now assume that in this way we have obtained six traces (Figure 1b), and that t_3, t_5 and t_6 are identified as traces that relate to the situation of interest: abnormal job execution that leads to failure (F) or abortion (A). Thus, $\{t_3, t_5, t_6\}$ is our sample of historic event data for which query discovery is to be performed. Each trace contains a subsequence of a scheduling event of low or high priority (SchLow or SchHigh), an eviction event (E), a scheduling event of the same priority as the initial one (SchLow or SchHigh), and another eviction event (E). These commonalities can be captured by an swg-query by explicitly representing the eviction events by the type E, while the occurrence of the scheduling events is captured by two occurrences of the same variable x, i. e., a suitable swg-query would be $x\mathsf{E}x\mathsf{E}$. Since the semantics of swg-queries is based on subsequences, in between the types and variables of $x\mathsf{E}x\mathsf{E}$ also other types like U, Sus and Res may occur. Furthermore, the query matches the traces with global window size of 6. Note that the global window size w has to be at least 6 to describe t_5 and t_6, because from the first scheduling event to the second eviction event both traces consist of 6 events. As gap-size constraint we could choose any tuple of the form $c = ((0, i), (0, j), (0, k))$, where $i, k \in \mathbb{N}_{\geqslant 2} \cup \{\infty\}$ and $j \in \mathbb{N} \cup \{\infty\}$ (observe that in t_5 two events occur between the last scheduling and eviction event, and in t_6 two events occur between the first scheduling and eviction event; thus, any c with $i, k \geqslant 2$ is suitable here). The most restrictive choice that suitably describes our example is $w = 6$ and $c = ((0, 2), (0, 0), (0, 2))$.

Languages for complex event processing describe queries which correlate events to detect event patterns, using operators such as sequencing, conjunction, Kleene closure, negation and variables which may be bound to events in a stream [5, 10, 31]. However, the differences in semantics and expressiveness of different languages are not yet well-studied and the formalisation of event queries is an active area of research. Defining a query is usually assumed to be a manual step requiring expert knowledge. The iCEP System [24] and the IL-Miner [21] form an exception, but both are limited regarding their expressive power and scalability, among others. Furthermore, the complexity of the event mining problem is still open; it has not yet received an in-depth investigation from a theoretical perspective.

As illustrated in Example 1.1, we want to find situations of interest that occur in many traces of the given sample set of traces. At a first glance this seems to be a similar problem as solved in the context of frequent sequence mining. Algorithms in this research area as [2, 8, 9, 30] require the a-priori definition of event abstractions; in particular, fine-grained conditions to generalise individual events or to correlate several events are not discovered by these approaches.

In this paper, we try to overcome the issues raised above by proposing a basic language that supports sequencing, variables that range over single events, and different kinds of window size constraints, i.e. local gap-size constraints and a global window size. Hence our swg-queries are located at the core of complex event processing languages while based on classical concepts of theoretical computer science: subsequences and (string) patterns with variables. Subsequences have extensively been studied both in a purely combinatorial sense (in formal language theory, logic and combinatorics on words) and algorithmically (in string algorithms and bioinformatics); see the introductions of the recent papers [20, 11] for a comprehensive list of relevant pointers. Patterns with variables are introduced by Angluin [3] and they play a central role for inductive inference, in formal language theory and combinatorics on words (see [28, 23, 25]). These *Angluin-style* patterns are just strings with terminal symbols $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$ and variables $x_1, x_2, x_3, \ldots$, e. g., $p = x_1 \, \mathsf{a} \, x_1 \, \mathsf{b} \, x_2$, and they match exactly the strings that can be obtained by uniformly replacing the variables by some terminal strings (i. e., exactly the strings $u \, \mathsf{a} \, u \, \mathsf{b} \, v$ with $u, v \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots\}^*$ for the example pattern). Syntactically, our swg-queries are Angluin-style patterns, but with the semantics adapted to event streams: variables only range over single symbols; and the query matches if, after replacing the variables by events, it occurs as a subsequence that satisfies the window size and gap-size constraints.

Despite the fundamental semantic differences between swg-queries and Angluin-style patterns, it is possible to adapt concepts and algorithms from inductive inference of the so-called *pattern languages* that can be described by Angluin-style patterns. Most importantly, the classical concept of *descriptive patterns*, already introduced in [3] for Angluin-style patterns (see also [18, 19]), can be adapted to swg-queries. For a given sample of strings, a descriptive Angluin-style pattern can be computed by Shinohara's algorithm [27], which employs a rather simple and robust algorithmic idea (see also [14]). The main result of this paper is that Shinohara's algorithm can also be adapted for computing descriptive swg-queries, and thus to the case of swg-query discovery. Due to the semantic differences, this adaption is non-trivial and requires the development of suitable technical tools. Moreover, our variant of the algorithm has the following special features:

- it allows a *support threshold* (i. e., a lower bound on the number of matched elements of the sample),
- it can be parameterised by any class of queries that satisfies some mild closure properties (this is crucial for dealing with complexity issues, as explained further below),
- it can also be used in order to check whether a given query is descriptive.

The algorithm computes a descriptive query by performing a number of iterations that is bounded by a low-degree polynomial in the length of the query and the size of the sample. However, in each iteration, the algorithm calls a sub-routine that solves the *matching problem*: Decide whether a given query matches a given trace. Just like for Angluin-style patterns, matching for swg-queries is intractable; thus, it constitutes a complexity bottle-neck in our algorithm. However, as another similarity between Angluin-style patterns and our swg-queries (see [14]), the matching problem for swg-queries reduces to the problem of computing descriptive queries. This means that this complexity bottle-neck inevitably exists in any possible algorithm for computing descriptive swg-queries. Therefore, we perform a thorough complexity analysis of the matching problem for swg-queries (note that for Angluin-style patterns the complexity of the matching problem is well-understood (see [15, 16])). On the positive side, since our algorithm can be parameterised by any arbitrary class of queries (with some natural closure properties), restrictions of swg-queries that lead to a tractable matching problem also lead to tractable computation of descriptive queries via our algorithm.

Typical brute-force approaches to data mining tasks (e. g., the well-known apriori-algorithm [1]) produce the full set of all objects of interest, but operate in exponential running time. Our algorithm produces only one descriptive query, but, for classes of queries with tractable matching problem, is rather fast. Moreover, by considering all possible runs of the algorithm, we can produce a large class of descriptive queries, although not all of them. We further illustrate the feasibility of our approach by developing a prototypical implementation of our algorithm and a performing a preliminary experimental study on a real-world dataset of cluster monitoring traces at Google [26].

The remainder of this paper is structured as follows. In Section 2, we propose swg-queries as an adaption of Angluin-style patterns to event streams, and we develop some technical tools for this query class. In Section 3, we present the algorithm, based on Shinohara's work on discovering descriptive patterns, which can be used (a) for discovering a descriptive swg-query from a given sample, and (b) for checking whether a given swg-query is descriptive with respect to a sample. Motivated by the observation that computing descriptive queries reduces to the matching problem, and vice versa (this connection is discussed at the end of Section 3), we perform in Section 4 a thorough (classical and parameterised) complexity analysis of the matching problem. Our results point out for which classes of queries efficient computation of descriptive queries is possible, and for which classes this is an intractable problem. Finally, in Section 5, we report some experimental results obtained by applying a prototypical implementation to a real-world dataset, and we give some concluding remarks. Due to space restrictions many technical details had to be deferred to the paper's full version.

2 Traces and Queries

By $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{N}_{\geqslant 1}$ we denote the set of integers, non-negative integers, and positive integers, respectively. For $\ell \in \mathbb{N}$ we let $[\ell] = \{i \in \mathbb{N}_{\geqslant 1} \ : \ 1 \leqslant i \leqslant \ell\}$. For a non-empty set A we write A^* (and A^+) for the set of all strings (the set of all non-empty strings) built from symbols in A. By $|s|$ we denote the length of a string s, and for a position $i \in [|s|]$ we write $s[i]$ to denote the letter at position i in s. A *factor* of a string $s \in A^*$ is a string $t \in A^*$ such that s is of the form $s_1 t s_2$ for $s_1, s_2 \in A^*$.

An *embedding* is a mapping $e : [\ell] \to [n]$ with $\ell \leqslant n$ such that $i < j$ implies $e(i) < e(j)$ for all $i, j \in [\ell]$. Let s and t be two strings with $|s| \leqslant |t|$. We say that s is a *subsequence of t with embedding* $e : [|s|] \to [|t|]$, if e is an embedding and $s[i] = t[e(i)]$ for every $i \in [|s|]$. We write $s \preccurlyeq_e t$ to indicate that s is a subsequence of t with embedding e; and we write $s \preccurlyeq t$ to indicate that there exists an embedding e such that $s \preccurlyeq_e t$.

We model traces as finite, non-empty strings over some (finite or infinite) alphabet Γ of *types*. It will be reasonable to assume that $|\Gamma| \geqslant 2$. A *trace* (over Γ) is a string $t \in \Gamma^+$. We write *types*(t) for the set of types that occur in t.

This paper studies a basic class of event queries which we call *subsequence-queries with wildcards and gap-size constraints*, for short: *swg-queries*. For defining these queries, we fix a countably infinite set Vars of *variables*, and we will always assume that Vars is disjoint with the set Γ of considered types. An *swg-query* q (over Vars and Γ) is specified by a *query string* $s \in (\mathsf{Vars} \cup \Gamma)^+$, a *global window size* $w \in \mathbb{N}_{\geqslant 1} \cup \{\infty\}$ with $w \geqslant |s|$, and a tuple $c = (c_1, c_2, \ldots, c_{|s|-1})$ *of local gap-size constraints* (*for* $|s|$ *and* w), where $c_i = (c_i^-, c_i^+) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$, such that $c_i^- \leqslant c_i^+$ for every $i \in [|s|-1]$ and $|s| + \sum_{i=1}^{|s|} c_i^- \leqslant w$. We denote such queries in the form $q = (s, w, c)$. Note that setting all gap-size constraints of a swg-query q to $(0, \infty)$ corresponds to a query without gap-size constraints.

We write *types*(q) (or *types*(s)) and *vars*(q) (or *vars*(s)) for the set of types and the set of variables, respectively, that occur in q's query string s. A query q is called an (ℓ, w, c)*-query* (over Vars and Γ) if $q = (s, w, c)$ with $|s| = \ell$; the parameter ℓ will be called *string length.* We write $\mathcal{Q}$ to denote the class of all swg-queries.

The semantics of swg-queries is defined as follows: Each variable in s serves as a *wildcard* representing an arbitrary type. A query $q = (s, w, c)$ *matches in* a trace t if the wildcards in s can be replaced by types in such a way that the resulting string s' satisfies the following: t contains a factor t' of length at most w such that s' occurs as a subsequence in t' and for each $i < \ell := |s|$ the gap between $s'[i]$ and $s'[i{+}1]$ in t' has length at least c_i^- and at most c_i^+. I.e., t' is of the form $s'[1]\, g_1\, s'[2]\, g_2 \cdots g_{\ell-1}\, s'[\ell]$ and $c_i^- \leqslant |g_i| \leqslant c_i^+$ for all $i \in [\ell{-}1]$.

An alternative description of these semantics, which will be more convenient for our formal proofs, involves a bit more notation: We say that an embedding $e : [\ell] \to [n]$ *satisfies* a global window size w, if $e(\ell) - e(1) + 1 \leqslant w$; and we say that e *satisfies* a tuple $c = (c_1, c_2, \ldots, c_{\ell-1})$ of local gap-size constraints (for ℓ and w), if $c_i^- \leqslant e(i{+}1) - 1 - e(i) \leqslant c_i^+$ for all $i < \ell$.

A *substitution* is a mapping $\mu : (\mathsf{Vars} \cup \Gamma) \to (\mathsf{Vars} \cup \Gamma)$ with $\mu(\gamma) = \gamma$ for all $\gamma \in \Gamma$. We extend substitutions to mappings $(\mathsf{Vars} \cup \Gamma)^+ \to (\mathsf{Vars} \cup \Gamma)^+$ in the obvious way, i. e., $\mu(s) = \mu(s[1])\mu(s[2]) \cdots \mu(s[\ell])$ for $s \in (\mathsf{Vars} \cup \Gamma)^+$ and $\ell := |s|$. We sometimes also consider partial substitutions of the form $(V \cup \Delta) \to (V' \cup \Delta)$ for some $V, V' \subseteq \mathsf{Vars}$ and $\Delta \subseteq \Gamma$.

An swg-query $q = (s, w, c)$ *matches in* a trace $t \in \Gamma^+$ (or, t *matches* q, in symbols: $t \models q$), if and only if there are a substitution $\mu : (\mathsf{Vars} \cup \Gamma) \to \Gamma$ and an embedding $e : [|s|] \to [|t|]$ that satisfies w and c, such that $\mu(s) \preccurlyeq_e t$. We call (μ, e) *a witness for* $t \models q$.

▶ **Example 2.1.** Let $x_1, x_2, x_3 \in \mathsf{Vars}$ and $\Gamma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. We consider a query $q = (s, w, c)$, where $s = x_1\, \mathsf{a}\, \mathsf{b}\, x_1\, \mathsf{b}\, x_2\, \mathsf{c}\, x_3\, \mathsf{a}\, x_1$, $w = 25$ and $c = ((0,1), (0,0), (2,\infty), (4,\infty),\ (0,5),\ (0,5),\ (0,5),\ (1,5), (2,3))$, and a trace $t = t_1\, \mathsf{c\,a\,b\,b\,b\,c\,a\,b\,a\,c\,a\,b\,a\,c\,b\,c\,a\,b\,a\,c}\, t_2$, where $t_1, t_2 \in \Gamma^*$. We observe that $t \models q$, and a witness substitution and embedding can be illustrated as follows:

$$\begin{array}{ccccccccccccccccc}
s & = & & x_1 & \mathsf{a\,b} & & x_1 & & \mathsf{b} & x_2 & \mathsf{c} & x_3 & & \mathsf{a} & & x_1\,, & \\
t & = & t_1 & \mathsf{c} & \mathsf{a\,b} & \mathsf{b\,b} & \mathsf{c} & \mathsf{a\,b\,a\,c\,a} & \mathsf{b} & \mathsf{a} & \mathsf{c} & \mathsf{b} & \mathsf{c} & \mathsf{a} & \mathsf{b\,a} & \mathsf{c} & t_2\,.
\end{array}$$

Recall that in the introduction we have presented another, more practical scenario that can also be described by an swg-query (see Example 1.1).

The *model set* of a query q w.r.t. a type set $\Delta \subseteq \Gamma$ is $\mathsf{Mod}_\Delta(q) \;:=\; \{\, t \in \Delta^+ \;:\; t \models q \,\}$. For two queries q and q' and a set $\Delta \subseteq \Gamma$ we say that q *is contained in* q' w.r.t. Δ if $\mathsf{Mod}_\Delta(q) \subseteq \mathsf{Mod}_\Delta(q')$; and we say that q *and* q' *are equivalent w.r.t.* Δ if $\mathsf{Mod}_\Delta(q) = \mathsf{Mod}_\Delta(q')$. The remainder of this section is devoted to a characterisation of containment and equivalence of (ℓ, w, c)-queries via suitable notions of homomorphisms and isomorphisms. Let us fix an $\ell \in \mathbb{N}_{\geqslant 1}$, a $w \in \mathbb{N} \cup \{\infty\}$ with $w \geqslant \ell$, and a tuple c of local gap-size constraints for ℓ and w.

▶ **Definition 2.2.** A *homomorphism* from an (ℓ, w, c)-query $q' = (s', w, c)$ to an (ℓ, w, c)-query $q = (s, w, c)$ is a substitution $h : (\mathsf{Vars} \cup \Gamma) \to (\mathsf{Vars} \cup \Gamma)$ such that $h(s') = s$ (i.e., $h(s'[i]) = s[i]$ for all $i \in [\ell]$). We write $q' \xrightarrow{\text{hom}} q$ to express that there exists a homomorphism from q' to q.

We next characterise containment via homomorphisms (see item (c) of the next theorem).

▶ **Theorem 2.3.** *Let q and q' be (ℓ, w, c)-queries over* Vars *and* Γ.

(a) *If $q' \xrightarrow{\text{hom}} q$ then $types(q') \subseteq types(q)$ and $\mathsf{Mod}_\Gamma(q') \supseteq \mathsf{Mod}_\Gamma(q)$.*

(b) *Let $\Delta \subseteq \Gamma$ be such that $|\Delta| \geqslant 2$ and $\Delta \supseteq types(q)$. If $\mathsf{Mod}_\Delta(q') \supseteq \mathsf{Mod}_\Delta(q)$ then $q' \xrightarrow{\text{hom}} q$.*

(c) *If $|\Gamma| \geqslant 2$, then $\mathsf{Mod}_\Gamma(q') \supseteq \mathsf{Mod}_\Gamma(q) \iff q' \xrightarrow{\text{hom}} q$.*

Let us consider an example application of this characterisation of containment.

▶ **Example 2.4.** Let us fix ℓ distinct variables $x_1, \ldots, x_\ell \in$ Vars, and let $s_{mg} := x_1 x_2 \cdots x_\ell$. The query $q_{mg} = (s_{mg}, w, c)$ is the *most general (ℓ, w, c)-query* in the sense that $\mathsf{Mod}_\Gamma(q_{mg}) \supseteq \mathsf{Mod}_\Gamma(q)$ for every (ℓ, w, c)-query $q = (s, w, c)$. To see this, note that the mapping defined via $h(x_i) := s[i]$ for all $i \in [\ell]$ provides a homomorphism from q_{mg} to q and use Theorem 2.3(a).

One might ask why Theorem 2.3 restricts attention to queries q and q' of the same parameters (ℓ, w, c). One answer is that this is exactly what is needed for our purposes in Section 3: to ensure that our algorithm for query discovery always produces *meaningful* results, we precisely need this characterisation (see also Remark 3.2). Another answer is that, when considering queries q and q' with different parameters (ℓ, w, c) and (ℓ', w', c'), a characterisation analogous to the one provided by Theorem 2.3(c) simply is not available:

▶ **Example 2.5.** For the sake of this example, assume that Definition 2.2 applies to arbitrary queries $q = (s, w, c)$ and $q' = (s', w', c')$ of the same query string length $\ell := |s| = |s'|$ but where the global window sizes and the local gap-size constrains may differ from each other.

First consider two swg-queries $q = (s, w, c)$ and $q' = (s, w, c')$ having the same query string $s = x\,\mathsf{a}\,x$ and the same global window size $w = 4$, but different local gap-size constraints $c = ((0, 1), (0, 0))$ and $c' = ((0, 0), (0, 0))$. It holds that $q' \xrightarrow{\text{hom}} q$, but $\mathsf{Mod}_\Gamma(q') \not\supseteq \mathsf{Mod}_\Gamma(q)$, e. g., $\mathsf{bcab} \in \mathsf{Mod}_\Gamma(q) \setminus \mathsf{Mod}_\Gamma(q')$.

Almost the same example queries show that the characterisation does also not hold for queries with the same query string length and the same gap-size constraints but different global window sizes: Let $q = (s, w, c)$ and $q' = (s, w', c)$ with $s = x\,\mathsf{a}\,x$, $c = ((0, \infty), (0, \infty))$, and $w = 4$ and $w' = 3$. Again, we can observe that $q' \xrightarrow{\text{hom}} q$, but $\mathsf{Mod}_\Gamma(q') \not\supseteq \mathsf{Mod}_\Gamma(q)$ (witnessed by the same trace bcab). Note furthermore, that tuples of gap-size constraints of the form $((0, \infty), \ldots, (0, \infty))$ can even be considered as not having any gap-size constraints.

There are natural candidates of extending the concept of homomorphisms to the case of queries with different query string lengths. For example, we could adapt Definition 2.2 by requiring the substitution $h : (\mathsf{Vars} \cup \Gamma) \to (\mathsf{Vars} \cup \Gamma)$ to satisfy that $h(s')$ is a subsequence of s. However, it can be easily seen that for $q := (s, w, c)$ and $q' := (s', w', c')$ with $s = x\,\mathsf{a}\,\mathsf{b}\,x$, $s' = x\,\mathsf{a}\,x$, $w = 4$, $w' = 3$, $c = ((0, \infty), (0, \infty), (0, \infty))$, and $c' = ((0, \infty), (0, \infty))$, we have $q' \xrightarrow{\text{hom}} q$ (with respect to the adapted definition), but $\mathsf{c\,a\,b\,c} \in \mathsf{Mod}_\Gamma(q) \setminus \mathsf{Mod}_\Gamma(q')$.

▶ **Definition 2.6.** Two (ℓ, w, c)-queries $q = (s, w, c)$ and $q' = (s', w, c)$ are called *isomorphic* (denoted by $q \cong q'$), if there is a bijection $\pi : (vars(q) \cup \Gamma) \to (vars(q') \cup \Gamma)$ such that $\pi(\gamma) = \gamma$ for all $\gamma \in \Gamma$ and $\pi(s[i]) = s'[i]$ for all $i \in [\ell]$.

▶ **Corollary 2.7.** *Let $|\Gamma| \geqslant 2$. For all (ℓ, w, c)-queries q and q' over Γ and* Vars *we have:*
$q \cong q' \iff \left(q \xrightarrow{\text{hom}} q' \text{ and } q' \xrightarrow{\text{hom}} q \right) \iff \mathsf{Mod}_\Gamma(q) = \mathsf{Mod}_\Gamma(q')$.
Furthermore, upon input of q and q' we can decide in time $\mathrm{O}(\ell)$ whether or not $q \cong q'$.

The following notion of "partially isomorphic" queries will be useful for Section 3.

▶ **Definition 2.8.** Let $q = (s, w, c)$ and $q' = (s', w, c)$ be (ℓ, w, c)-queries and let $I \subseteq [\ell]$. We say that *q is partially isomorphic to q' w.r.t. I* (and shortly write $q \sim_I q'$) if, and only if,

1. for all $i, j \in I$ we have $\left(s[i] = s[j] \iff s'[i] = s'[j] \right)$, and
2. for all $i \in I$ we have $\left(s[i] \in \Gamma \iff s'[i] \in \Gamma \right)$ and $\left(\text{if } s[i] \in \Gamma \text{ then } s[i] = s'[i] \right)$.

▶ **Lemma 2.9.** *For all (ℓ, w, c)-queries q and q' we have: $q \sim_{[\ell]} q' \iff q \cong q'$.*

3 Discovery of Descriptive (ℓ, w, c)-Queries

A *sample* is a finite, non-empty set $\mathcal{S}$ of traces over Γ. The *support* $\mathsf{supp}(q, \mathcal{S})$ of a query q in $\mathcal{S}$ is defined as the fraction of traces in the sample that match q, i.e., $\mathsf{supp}(q, \mathcal{S}) := \frac{|\{t \in \mathcal{S} \,:\, t \models q\}|}{|\mathcal{S}|}$. A *support threshold* is a rational number sp with $0 < \mathsf{sp} \leqslant 1$. A query q is said to *cover* a sample $\mathcal{S}$ *with support* sp if $\mathsf{supp}(q, \mathcal{S}) \geqslant \mathsf{sp}$.

▶ **Definition 3.1.** Let Q be a class of queries, let $\mathcal{S}$ be a sample, and let sp be a support threshold. Let $\ell \in \mathbb{N}_{\geqslant 1}$, let $w \in \mathbb{N} \cup \{\infty\}$ with $w \geqslant \ell$, and let c be a tuple of local gap-size constraints for ℓ and w. A query q is called *descriptive for* $\mathcal{S}$ *w.r.t.* $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ if q is an (ℓ, w, c)-query in Q, $\mathsf{supp}(q, \mathcal{S}) \geqslant \mathsf{sp}$, and there is no (ℓ, w, c)-query $q' \in \mathsf{Q}$ with $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ and $\mathsf{Mod}_\Gamma(q') \subsetneq \mathsf{Mod}_\Gamma(q)$.

Let us explain this notion on an intuitive level by considering the case $\mathsf{sp} = 1$. In this case, a query q that is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ satisfies $\mathcal{S} \subseteq \mathsf{Mod}_\Gamma(q)$, and there is no other (ℓ, w, c)-query $q' \in \mathsf{Q}$ that can be "wrapped around" $\mathcal{S}$ more tightly, i. e., in the sense that $\mathcal{S} \subseteq \mathsf{Mod}_\Gamma(q') \subsetneq \mathsf{Mod}_\Gamma(q)$. Therefore, among all (ℓ, w, c)-queries from Q, the query q can be considered as one of the best descriptors for $\mathcal{S}$.

▶ Remark 3.2. Throughout this section we assume that $|\Gamma| \geqslant 2$. Hence, by Theorem 2.3 and Corollary 2.7 we know that, for any query class $\mathsf{Q} \subseteq \mathcal{Q}$, an swg-query q is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ if, and only if, q is an (ℓ, w, c)-query in Q, $\mathsf{supp}(q, \mathcal{S}) \geqslant \mathsf{sp}$, and there is no (ℓ, w, c)-query $q' \in \mathsf{Q}$ with $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ and $q \xrightarrow{\text{hom}} q'$ and $q \not\cong q'$.

The following notions will be convenient throughout the rest of this section. For an (ℓ, w, c)-query $q = (s, w, c)$ and a symbol $z \in \mathsf{Vars} \cup \Gamma$ we let $pos(q, z)$ (and $pos(s, z)$) be the set of all positions i of s that carry the symbol z. I.e., $pos(q, z) = pos(s, z) = \{i \in [\ell] : s[i] = z\}$. Given a query string $s \in (\mathsf{Vars} \cup \Gamma)^\ell$, a set of positions $P \subseteq [\ell]$, and a symbol $z \in \mathsf{Vars} \cup \Gamma$, we write $s\langle P \mapsto z\rangle$ to denote the query string s' obtained from s by placing z on all positions $i \in P$. I.e., $s'[i] = z$ for all $i \in P$ and $s'[j] = s[j]$ for all $j \in [\ell] \setminus P$. Accordingly, for a query $q = (s, w, c)$ we let $q\langle P \mapsto z\rangle$ be the query $(s\langle P \mapsto z\rangle, w, c)$. For a variable $x \in vars(q)$ we shortly write $q\langle x \mapsto z\rangle$ (and $s\langle x \mapsto z\rangle$) instead of $q\langle pos(q, x) \mapsto z\rangle$ (and $s\langle pos(s, x) \mapsto z\rangle$) – i.e., all occurrences of variable x in s are replaced by the symbol z.

Next, we define the problem of computing a descriptive query, and the problem to check for a given query whether it is descriptive; we will refer to these two problems as the *query discovery problems*. As our definition of descriptiveness (see Definition 3.1) depends on a class Q of swg-queries, the problems of computing a descriptive query or of checking whether a given query is descriptive are also parameterised by an arbitrary class Q of swg-queries. It will be discussed in more detail in Section 4 why this parameterisation by Q is vital.

Q-Compute Descriptive Query Problem (Q-CompDescQuery): The input is a sample $\mathcal{S}$ over Γ, a support threshold sp, a string length $\ell \in \mathbb{N}$, a global window size $w \geqslant \ell$, and a tuple c of local gap-size constraints (for ℓ and w). The task is to compute an (ℓ, w, c)-query $q \in \mathsf{Q}$ that is descriptive for $\mathcal{S}$ with respect to $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$.

Q-Check Descriptiveness Problem (Q-CheckDescQuery): The input is the same as for Q-CompDescQuery, but in addition also an (ℓ, w, c)-query $q \in \mathsf{Q}$, and the task is to decide whether q is descriptive for $\mathcal{S}$ with respect to $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$.

▶ Remark 3.3. Since a Q-CompDescQuery- or Q-CheckDescQuery-instance contains at most $2(|s|-1)+1$ integers (i. e., w and c_i^-, c_i^+ for every $i \in [|s|-1]$), running times polynomial in w, c_i^+ or c_i^- can be exponential in the input size. We deal with this issue by assuming, without loss of generality, that $w = \infty$ or $w \leqslant \max\{|t| \colon t \in \mathcal{S}\}$, $c_i^+ = \infty$ or $c_i^+ \leqslant \max\{|t| \colon t \in \mathcal{S}\}$ for every $i \in [|s|-1]$, and $\sum_{i=1}^{\ell} c_i^- \leqslant \min\{w, \max\{|t| \colon t \in \mathcal{S}\}\}$. I.e., all integers of the input are bounded by $\max\{|t| \colon t \in \mathcal{S}\}$ and therefore their values are polynomial in the total input size.

3.1 An Algorithm for Solving the Query Discovery Problems

We now present an algorithm that, for suitable classes Q of queries, can be used to solve the problems Q-$\mathsf{CompDescQuery}$ and Q-$\mathsf{CheckDescQuery}$. As input, the algorithm receives $\mathcal{S}$, sp, (ℓ, w, c), and an "initial" query $q_0 = (s_0, w, c) \in \mathsf{Q}$. The goal is to compute an (ℓ, w, c)-query $q \in \mathsf{Q}$ that is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ and that satisfies $\mathsf{Mod}_\Gamma(q) \subseteq \mathsf{Mod}_\Gamma(q_0)$.

If $\mathsf{supp}(q_0, \mathcal{S}) < \mathsf{sp}$, then such a query q does not exist because of the following reasoning: obviously, if q_0 does not describe $\mathcal{S}$ with sufficient support, then no other query q' with $\mathsf{Mod}_\Gamma(q') \subseteq \mathsf{Mod}_\Gamma(q_0)$ can describe $\mathcal{S}$ with sufficient support.

If $\mathsf{supp}(q_0, \mathcal{S}) \geqslant \mathsf{sp}$, then the algorithm proceeds as follows. It considers the variables $\mathit{vars}(q_0) = \{x_1, x_2, \ldots, x_k\}$ in some arbitrary order. For each x_i, it performs the following *replacement operation*: choose a symbol y (which can be a type or a variable) and replace *all* occurrences of x_i in the current query string by y. However, such a replacement operation is only admissible if the new query q' is in Q and satisfies $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$. If no admissible replacement operation is possible, then x_i is not replaced, i. e., the current query string is not changed. After each variable $x_i \in \mathit{vars}(q_0)$ has been considered, the algorithm terminates and produces the current query as output.

This means that the algorithm produces a sequence $s_0, s_1, \ldots, s_k$ of query strings, where, for every $i \in [k]$, either s_i can be obtained from s_{i-1} by a replacement operation, or $s_i = s_{i-1}$. For every $i \in [k]$, let $q_i = (s_i, w, c)$ be the (ℓ, w, c)-query that corresponds to s_i. We note that $q_0 \xrightarrow{\text{hom}} q_1 \xrightarrow{\text{hom}} \ldots \xrightarrow{\text{hom}} q_k$ and therefore, as a consequence of Theorem 2.3, $\mathsf{Mod}_\Gamma(q_0) \supseteq \mathsf{Mod}_\Gamma(q_1) \supseteq \ldots \supseteq \mathsf{Mod}_\Gamma(q_k)$. Hence, the algorithm starts with q_0 and then follows a path of length at most k in the search space of (ℓ, w, c)-queries from Q that cover $\mathcal{S}$ with support sp, always going from one query to a more specific one. The crucial point is now that if the replacement operations are done in a certain way, then it can be guaranteed that the final query q_k is necessarily *descriptive* for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. In other words, from any (ℓ, w, c)-query $q \in \mathsf{Q}$ that covers our sample with sufficient support, we will reach with at most $|\mathit{vars}(q_0)|$ replacement operations a descriptive query.

Let us call a replacement operation a *type operation* or a *variable operation* if it replaces x_i by a type or by a variable, respectively. For simplicity, assume that the algorithm considers (and possibly replaces) the variables in the order $x_1, x_2, \ldots, x_k$. Assume that we are at the i^{th} step of the algorithm, i. e., variable x_i is considered and the goal is to obtain s_i from s_{i-1}. The symbol y we choose to replace for x_i can either be a type or a variable. For a type operation we can choose from all types that occur in sufficiently many traces of the sample $\mathcal{S}$. For a variable operation we can choose one of the variables x_j that have been considered before (i. e., $j < i$) and that have *not* been replaced (i. e., at the j^{th} step when we considered x_j, no replacement operation was admissible). Let us clarify this with an example.

Let $s_0 = x_2 x_1 \, \mathsf{a} \, x_1 \, \mathsf{b} \, x_3 x_4$, and let us assume that we first consider x_1. With the restriction mentioned above, we can initially only use type operations. For the sake of the example, assume further that no type operation for x_1 is admissible (since it would yield $q_1 \notin \mathsf{Q}$ or $\mathsf{supp}(q_1, \mathcal{S}) < \mathsf{sp}$); thus, $s_1 = s_0$. Next we consider variable x_2 and we note that since x_1 has already been considered and has not been replaced in the first step, $y := x_1$ is a valid candidate for a variable operation. Let us assume that it turns out that it also is admissible. Then, we can replace x_2 by x_1 and obtain $s_2 = x_1 x_1 \, \mathsf{a} \, x_1 \, \mathsf{b} \, x_3 x_4$. However, this also means that x_2 will never be available as a valid candidate for variable operations later on (which is also the case if a type operation had been used for replacing x_2 with an admissible type).

To see that this restriction is indeed necessary, assume the following example: The sample $\mathcal{S}$ consists of only one trace, namely the trace aaa, and we have ℓ=3, w=3, $c = ((0,0),(0,0))$, and $s_0 = x_1x_2x_3$. Without the restriction from above, we could obtain $s_3 = x_1x_1x_1$ with variable operations by replacing x_1, x_2 and x_3 by the same variable x_1. Note that replacing these variables by x_1 indeed violates our restriction, because when considering x_1 in the first step, replacing x_1 by the type a is in fact an admissible replacement operation and therefore we *have* to replace x_1 in the algorithm's first step (and hence in later steps the variable x_1 is not available). If we perform the replacement operations according to our restrictions, we in fact perform the type operation with $y := \mathsf{a}$ for each of the variables x_1, x_2, x_3, resulting in the query $q' = (\mathsf{aaa}, w, c)$. Note that the query $q = (x_1x_1x_1, w, c)$, which could be otained when ignoring our restriction, is *not* descriptive (since $\mathsf{Mod}_\Gamma(q') \subsetneq \mathsf{Mod}_\Gamma(q)$). This illustrates that we indeed need the restriction described above. Note that q' is actually the only descriptive query for the particular example considered here.

With the restricted kind of variable operations, we can show that the algorithm always produces descriptive queries, provided that the query class Q has suitable closure properties. Pseudocode of the algorithm is provided in Algorithm 1. Let us briefly explain how the pseudocode actually implements the idea outlined above. Initially, we collect in Δ the types available for type operations (note that if $\frac{|\{t \in \mathcal{S} \,:\, \gamma \in types(t)\}|}{|\mathcal{S}|} < \mathsf{sp}$, then the type γ cannot occur in any query that covers $\mathcal{S}$ with support sp), we initialise the set U of *unvisited* variables as the set of *all* variables of s_0, and we define $V := \emptyset$ as the (empty) set of variables that are initially available for variable operations. Now the main loop in line 5 considers each variable $x \in U$ exactly once, and for each such variable, checks whether there exists a type or variable y from $\Omega := (\Delta \cup V)$ such that replacing all occurrences of variable x by the symbol y is an admissible replacement operation. If such an $y \in \Omega$ exists, we perform the actual replacement in line 12. If no such element in Ω exists, then no replacement operation is possible, and we do not change the current query string but insert x into the set V of variables available for future variable operations (line 16).

Theorem 3.4 summarises the guarantees we can provide for Algorithm 1. For the precise statement we need two notions of closure properties that the query class Q has to satisfy: Q is called *closed under isomorphisms* if for all parameters (ℓ, w, c) and for every (ℓ, w, c)-query $q \in \mathsf{Q}$, the set Q also contains all (ℓ, w, c)-queries q' with $q \cong q'$. Q is called *closed under m-grained generalisations*, for an $m \in \mathbb{N}_{\geqslant 1} \cup \{\infty\}$, if the following is true for every query $q = (s, w, c) \in \mathsf{Q}$ and every set of positions $P \subseteq [|s|]$ with $|P| \leqslant m$ and $s[i] = s[j]$ for all $i, j \in P$: There is a variable $x \in \mathsf{Vars} \setminus vars(q)$ such that the query $q\langle P \mapsto x\rangle$ belongs to Q.

We explain this concept on an intuitive level by considering $m = 1$. A 1-grained generalisation takes a $q \in \mathsf{Q}$ and makes it more general by replacing the symbol at some position of the query string by a single occurrence of a new variable. E. g., $x\,\mathsf{a}\,yx\,\mathsf{b} \rightsquigarrow xz_1yx\,\mathsf{b} \rightsquigarrow xz_1z_2x\,\mathsf{b} \rightsquigarrow z_3z_1z_2x\,\mathsf{b}$ is a sequence of 1-grained generalisations, where z_1, z_2, z_3 are new variables. If for every $q \in \mathsf{Q}$ every 1-grained generalisation necessarily produces a query in Q, then Q is closed under 1-grained generalisations. For example, this is the case for the class $\mathcal{Q}^{|rv(s)| \leqslant k}$ of queries with at most k repeated variables, i.e. variables with at least two occurrences in s, or the class $\mathcal{Q}^{w \leqslant k}$ of queries with window size at most k, for every constant $k \in \mathbb{N}$. On the other hand, the class $\mathcal{Q}^{|vars(s)| \leqslant k}$ of queries with at most k variables is not closed under 1-grained generalisations. An m-grained generalisation replaces in one step at most m occurrences of *the same* symbol by occurrences of the same new variable. Since m-grained generalisations for $m \geqslant 2$ introduce new repeated variables, the class $\mathcal{Q}^{|rv(s)| \leqslant k}$ is not closed under m-grained generalisations. On the other hand, the class $\mathcal{Q}^{|s| \leqslant k}$, which consists only of queries with a query string length of at most k, is closed under

Algorithm 1 Q-DescrSWGQuery($\mathcal{S}$,sp, q_0).

```
   Input    : sample $\mathcal{S}$; support threshold sp with $0 < \mathsf{sp} \leqslant 1$; query $q_0 = (s_0, w, c)$ in Q
   Returns : descriptive query $q$ for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (|s_0|, w, c))$ or error message $\bot$
 1 if supp($q_0$, $\mathcal{S}$) < sp then stop and return $\bot$
 2 $\Delta := \{\gamma \in \Gamma \ : \ \frac{|\{t \in \mathcal{S} \ : \ \gamma \in types(t)\}|}{|\mathcal{S}|} \geqslant \mathsf{sp}\}$          // types to be considered
 3 $s := s_0$;  $q := (s, w, c)$                                  // query string and query
 4 $U := vars(q_0)$;  $V := \emptyset$                  // unvisited variables and available variables
 5 while $U \neq \emptyset$ do
 6 |   select an arbitrary $x \in U$ and let $U := U \setminus \{x\}$
 7 |   $\Omega := (\Delta \cup V)$;  replace := False
 8 |   while $\Omega \neq \emptyset$ do
 9 |   |   select an arbitrary $y \in \Omega$ and let $\Omega := \Omega \setminus \{y\}$
10 |   |   $q' := (s\langle x \mapsto y\rangle, w, c)$
11 |   |   if $q' \in \mathsf{Q}$ and supp($q'$, $\mathcal{S}$) $\geqslant$ sp then
12 |   |   |   $s := s\langle x \mapsto y\rangle$                                    // ReplaceOp
13 |   |   |   replace := True
14 |   |   |   break inner loop
15 |   if replace is False then
16 |   |   $V := V \cup \{x\}$                                        // NoChangeOp
17 stop and return $q := (s, w, c)$
```

∞-grained generalisations. Since $q' \xrightarrow{\text{hom}} q$ for every query q' obtained from q by an m-grained generalisation, Theorem 2.3(a) implies that q' is actually a more general query in the sense that $\mathsf{Mod}_\Gamma(q') \supseteq \mathsf{Mod}_\Gamma(q)$.

We are now ready for the formal statement of this subsection's main theorem. Note that the lines 6 and 9 of Algorithm 1 allow to make an arbitrary choice. Different choices lead to different "runs" of the algorithm, and different runs might produce different output queries (depending on the particular order in which the elements in $vars(q_0)$ and in Ω are considered). The next theorem states that *any* choice is fine, and that *any* output query is guaranteed to be a descriptive query.

▶ **Theorem 3.4.** *Let $|\Gamma| \geqslant 2$ and let* Q *be a class of swg-queries over Γ and* Vars *that is closed under isomorphisms. Let $\mathcal{S}$ be a sample, let* sp *be a support threshold, and let $q_0 = (s_0, w, c)$ be a query in* Q*. Let $\ell = |s_0|$ and let $m \in \mathbb{N}$ be such that $|pos(q_0, x)| \leqslant m$ for all $x \in vars(q_0)$.*

(a) *If $\mathsf{supp}(q_0, \mathcal{S}) < \mathsf{sp}$, then there does not exist any descriptive query q' for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ such that $\mathsf{Mod}_\Gamma(q') \subseteq \mathsf{Mod}_\Gamma(q_0)$, and there is only one run of algorithm* Q-DescrSWGQuery($\mathcal{S}$,sp, q_0)*, and this run stops in line 1 with output $\bot$.*

(b) *If $\mathsf{supp}(q_0, \mathcal{S}) \geqslant \mathsf{sp}$ and* Q *is closed under m-grained generalisations, then every run of* Q-DescrSWGQuery($\mathcal{S}$,sp, q_0) *terminates and outputs a query q' that is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ and satisfies $\mathsf{Mod}_\Gamma(q') \subseteq \mathsf{Mod}_\Gamma(q_0)$; furthermore, $q' = q_0$ if, and only if, q_0 is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$.*

The proof of (a) is obvious while the proof of (b) is quite demanding and proceeds as follows. For simplicity, let us focus here on the case that $\mathsf{Q} = \mathcal{Q}$. We have already observed above that the output query $q' := q_k$ is necessarily an (ℓ, w, c)-query from Q with $\mathsf{supp}(q_k, \mathcal{S}) \geqslant \mathsf{sp}$. Now, for contradiction, let us assume that q_k is not descriptive for $\mathcal{S}$ with respect to $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. Then, according to Definition 3.1, there exists an (ℓ, w, c)-query $\tilde{q} \in \mathsf{Q}$ with support $\mathsf{supp}(\tilde{q}, \mathcal{S}) \geqslant \mathsf{sp}$ and $\mathsf{Mod}_\Gamma(\tilde{q}) \subsetneq \mathsf{Mod}_\Gamma(q_k)$. This means that $q_k \xrightarrow{\text{hom}} \tilde{q}$ and $\tilde{q} \not\cong q_k$ (see

Remark 3.2). We can now inductively show that for every $i \in \{0, \dots, k\}$ the queries q_i and $\tilde{q}$ are partially isomorphic (see Definition 2.8) with respect to the positions of the variables already considered by the algorithm. For $i = k$ this yields $\tilde{q} \cong q_k$; a contradiction. Proving the induction step is non-trivial and constitutes the most crucial technical contribution of the correctness proof. In particular, we heavily rely on the technical machinery developed in Section 2. A detailed proof will be included in the full version of this paper.

We now discuss how the algorithm can be used to solve the problems Q-$\mathsf{CompDescQuery}$ and Q-$\mathsf{CheckDescQuery}$. For solving the problem Q-$\mathsf{CompDescQuery}$, we only need that Q is closed under isomorphisms and 1-grained generalisations, since then, we can run it with $s_0 = x_1 x_2 \dots x_\ell$ (i. e., the most general (ℓ, w, c)-query, see Example 2.4) in order to compute a query that is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. On the other hand, if we want to solve Q-$\mathsf{CheckDescQuery}$ for some (ℓ, w, c)-query $q \in \mathsf{Q}$, then we can run the algorithm with $q_0 := q$, and then check whether its output equals q_0. The latter is the case if, and only if, q is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. However, this only works for queries $q = (s, w, c)$ such that Q is closed under m-grained generalisations, where $m \in \mathbb{N}$ is such that $|pos(q, x)| \leqslant m$ for all $x \in vars(q)$.

Let us close this subsection by an outlook on future work. Clearly, a single run of Algorithm 1 only outputs a single query that is descriptive w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. However, by exploring all possible runs it is possible to enumerate a large number of queries that are descriptive w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. But care must be taken when exploring the search space of all possible runs, since two different runs of Algorithm 1 may lead to the same output query and we want to avoid outputting the same query twice. We plan to address this enumeration task and suitable selection strategies in our future work. An obvious question is if *every* descriptive query can be produced by a suitable run of Algorithm 1. The answer is "no", as the following example shows. Given the sample $\mathcal{S} = \{\mathsf{aab}, \mathsf{abb}\}$, $\mathsf{sp} = 1$ and $q_0 = (x_1 x_2, w, c)$ with $w = 2$ and $c = ((0, \infty))$, *every* run of Algorithm 1 outputs the descriptive query $q := (\mathsf{ab}, w, c)$. But note that also the query $q' := (x_i x_i, w, c)$ (for an arbitrary $i \in \{1, 2\}$) is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (2, w, c))$. However, there does not exist any run of Algorithm 1 that outputs q'. Obvious future tasks are (a) to find a precise characterisation of the descriptive queries that can be produced by a run of Algorithm 1, and (b) to design an efficient algorithm that is capable of producing *all* descriptive queries.

3.2 The Complexity of the Query Discovery Problems

In Algorithm 1, we treat the checks of $q' \in \mathsf{Q}$ and $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ in line 11 as black-box requests to an oracle. Clearly, we can check if $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ by solving the following *matching problem* for $|\mathcal{S}|$ times. For any fixed query class $\mathsf{Q} \subseteq \mathcal{Q}$, the **Q-Matching Problem (Q-Match)** receives as input a query $q = (s, w, c) \in \mathsf{Q}$ and a trace t. The task is to decide if $t \models q$.

Since Q-Match-instances contain integers, we make, w.l.o.g., the same assumptions as described in Remark 3.3: we assume $w = \infty$ or $w \leqslant |t|$, $c_i^+ = \infty$ or $c_i^+ \leqslant |t|$ for every $i \in [|s|-1]$, and $\sum_{i=1}^{\ell} c_i^- \leqslant \min\{w, |t|\}$. For the following complexity analysis, let us call the oracles that check $q' \in \mathsf{Q}$ and solve Q-Match the Q-*membership oracle* and Q-*match oracle*, respectively. A straightforward analysis of Algorithm 1 yields the following.

▶ **Theorem 3.5.** *Every run of* Q-$\mathsf{DescrSWGQuery}(\mathcal{S}, \mathsf{sp}, q_0)$ *performs* $\mathrm{O}(|vars(q_0)|(|types(\mathcal{S})| + |vars(q_0)|)(\ell + |\mathcal{S}|))$ *computational steps,* $\mathrm{O}(|vars(q_0)|(|types(\mathcal{S})| + |vars(q_0)|))$ *calls to the* Q-*membership oracle and* $\mathrm{O}(|vars(q_0)|(|types(\mathcal{S})| + |vars(q_0)|)|\mathcal{S}|)$ *calls to the* Q-*match oracle.*

Theorem 3.5 shows that apart from answering the oracle requests, the complexity of algorithm Q-DescrSWGQuery is rather low. For most natural classes of queries Q, it is a reasonable assumption that checking $q \in \mathsf{Q}$ for abitrary queries $q \in \mathcal{Q}$ is a computationally simple task (i. e., solvable in time linear or quadratic in $|s|$). Hence, not much complexity seems to be hidden in the Q-membership oracle. On the other hand, as shall be discussed in Section 4, Q-Match is NP-complete for many classes Q; thus, substantial computational complexity is hidden in the Q-match oracle.

Upper bounds for Q-Match directly yield upper bounds for the query discovery problems via Theorem 3.5. On the other hand, the intractability of Q-Match is only an obstacle for efficient query discovery w.r.t. algorithms that are based on solving the matching problem (like Q-DescrSWGQuery). However, as we shall demonstrate next, the matching problem can be reduced in polynomial time to our query discovery problems; thus, *every* algorithm for one of the latter problems necessarily implicitly also solves the matching problem. This implies that lower bounds on the complexity of the matching problem carry over to lower bounds on the complexity of our query discovery problems. Consequently, the problem Q-Match (and its complexity) is central for query discovery.

The reduction from Q-Match to our query discovery problems is based on the following construction. Let $\mathsf{Q} \subseteq \mathcal{Q}$ be an arbitrary set of queries that is closed under isomorphisms and assume that $|\Gamma| \geqslant 2$. Let $q = (s, w, c) \in \mathsf{Q}$ be an (ℓ, w, c)-query with $vars(q) = \{x_1, x_2, \ldots, x_k\}$, and let t be a trace. Let $\delta, \delta' \in \Gamma$ with $\delta \neq \delta'$. For every $i \in [k]$, let μ_{x_i} be the substitution that maps x_i to δ and all other variables to δ'. Let $\mu_{\delta'}$ be the substitution defined by $\mu_{\delta'}(x_j) = \delta'$ for every $j \in [k]$. For every $z \in \{x_1, x_2, \ldots, x_k, \delta'\}$, let t_z be a trace over Γ defined by

$$t_z \ := \ \ \mu_z(s[1])\ g_1\ \mu_z(s[2])\ g_2\ \cdots\ \mu_z(s[\ell{-}1])\ g_{\ell-1}\ \mu_z(s[\ell])\,,$$

where, for every $i < \ell$, g_i is the "gap string" consisting of c_i^- copies of some type from Γ. We now define the sample $\mathcal{S}_{q,t} := \{t, t_{\delta'}, t_{x_1}, t_{x_2}, \ldots, t_{x_k}\}$. By construction, for every $t' \in \mathcal{S} \setminus \{t\}$, $t' \models q$ is witnessed by the same embedding. We can show the following.

▶ **Theorem 3.6.** *The following statements are equivalent:*

1. $t \models q$*,*
2. $q \cong q'$ *for every* (ℓ, w, c)*-query* q' *that is descriptive for* $\mathcal{S}_{q,t}$ *with respect to* $(\mathsf{Q}, 1, (\ell, w, c))$*,*
3. q *is descriptive for* $\mathcal{S}_{q,t}$ *with respect to* $(\mathsf{Q}, 1, (\ell, w, c))$*.*

Theorem 3.6 reduces Q-Match to Q-CompDescQuery and Q-CheckDescQuery as follows. We can check $t \models q$ for a given (ℓ, w, c)-query q and a trace t by computing an (ℓ, w, c)-query q' that is descriptive for $\mathcal{S}_{q,t}$ with respect to $(\mathsf{Q}, 1, (\ell, w, c))$ and then checking $q \cong q'$, or by checking whether q is descriptive for $\mathcal{S}_{q,t}$ with respect to $(\mathsf{Q}, 1, (\ell, w, c))$. The next section is devoted to a detailed study of the complexity of Q-Match.

4 The Complexity of the Matching Problem

The algorithm presented in Section 3 computes a query that is descriptive for a sample $\mathcal{S}$ with respect to $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. This algorithm is based on the matching problem, i. e., it needs a sub-routine that solves Q-Match: check whether $t \models q$ for a given query $q \in \mathsf{Q}$ and a given trace t. As discussed at the end of Section 3, the algorithm only performs a polynomial number of such calls to a Q-match oracle, so the only source of exponential complexity might be hidden in the sub-routine solving Q-Match. Moreover, as demonstrated by Theorem 3.6 and the explanations following this result, the potential intractability of Q-Match inevitably makes not only the algorithm presented in Section 3 inefficient, but *every* algorithm that computes descriptive queries. In this section we show that the matching problem for general swg-queries is NP-complete. In order to identify tractable subclasses, it therefore is absolutely vital that our algorithm from Section 3 can be parameterised by an arbitrary class Q of queries, since then it does not need to solve the matching problem for arbitrary queries, but only for queries from Q, for which the matching problem might be tractable.

In other words: From Section 3.2 we know that computing descriptive queries with respect to Q is tractable if, and only if, the matching problem for the class Q is tractable. Therefore, the complexity of the problem Q-Match is crucial for the complexity of computing descriptive queries. This section is devoted to a comprehensive classical and parameterised complexity analysis of the matching problem.

Recall from Section 2 that $\mathcal{Q}$ denotes the class of all swg-queries. The input of the matching problem consists of a swg-query $q = (s, w, c)$ and a trace t. We consider the parameters shown in Figure 2, where $repvars(q) = repvars(s)$ is the set of repeated variables (i. e., variables with at least two occurrences in s), $maxgaps(c) = \max\{c_i^+ - c_i^- + 1 \colon i \in [|s|-1]\}$ and $maxc^+(c) = \max\{c_i^+ \colon i \in [|s|-1]\}$. Whenever we consider w as a parameter, we will assume that $w \neq \infty$; and if we consider $maxgaps(c)$ or $maxc^+(c)$ as a parameter, we will assume that $c_i^+ \neq \infty$ for every $i \in [|s|-1]$.

$\lvert t\rvert$	length of the trace t,	$maxc^+(c)$	maximum upper local gap-size,
w	global window size,	$\lvert repvars(s)\rvert$	number of repeated variables in s,
$\lvert s\rvert$	length of query string,	$\lvert types(t)\rvert$	number of types in the trace t,
		$maxgaps(c)$	maximum gap size.

Figure 2 Parameters of the problem $\mathcal{Q}$-Match.

For studying restricted or parameterised variants of $\mathcal{Q}$-Match, we use the following terminology. Let $p_1, p_2, \ldots$ be some of the parameters defined above. By $\mathcal{Q}$-Match *parameterised by* $(p_1, p_2, \ldots)$, we denote the parameterised variant of $\mathcal{Q}$-Match that arises from considering $p_1, p_2, \ldots$ as parameters. Moreover, if we write $p_i \leqslant k$ for some parameter p_i and some constant $k \in \mathbb{N}$, then we consider the problem variant were parameter p_i is bounded by the constant k. If all mentioned parameters are bounded by a constant, then the resulting problem variant is not a parameterised problem and therefore we write $\mathcal{Q}$-Match *with* $(p_1 \leqslant k_1, p_2 \leqslant k_2, \ldots)$. For example, by $\mathcal{Q}$-Match *parameterised by* $(|s|, |types(t)|, maxgaps(c) \leqslant 3)$ we denote the parameterised variant of $\mathcal{Q}$-Match where $|s|$ and $|types(t)|$ are parameters, and inputs are restricted to instances satisfying $maxgaps(c) \leqslant 3$; and by $\mathcal{Q}$-Match *with* $(|types(t)| \leqslant 10, maxgaps(c) \leqslant 3)$ we denote the classical decision problem $\mathcal{Q}$-Match where the input is restricted to instances satisfying $|types(t)| \leqslant 10$ and $maxgaps(c) \leqslant 3$.

We start by studying the complexity of the classical (non-parameterised) problems: For all subsets $p_1, p_2, \ldots, p_r$ of the parameters defined above and for all constants $k_1, k_2, \ldots, k_r \in \mathbb{N}$, we aim at answering the following question:

▶ **Question 4.1.** *Is* $\mathcal{Q}$-Match *with* $(p_1 \leqslant k_1, p_2 \leqslant k_2, \ldots, p_r \leqslant k_r)$ *in* P*, or is it* NP*-hard?*

With respect to parameters $|t|$, w, $|s|$, and $|repvars(s)|$, answers to Question 4.1 can be obtained with moderate effort:

▶ **Theorem 4.2.** *For every constant* $k \in \mathbb{N}$*, each of the following problems is in* P*:* $\mathcal{Q}$-Match *with* $|t| \leqslant k$*,* $\mathcal{Q}$-Match *with* $w \leqslant k$*,* $\mathcal{Q}$-Match *with* $|s| \leqslant k$*,* $\mathcal{Q}$-Match *with* $|repvars(s)| \leqslant k$*.*

Proof sketch. A natural brute-force approach is as follows: Upon input of an swg-query $q = (s, w, c)$ and a trace t, we enumerate all mappings $\pi : \mathit{repvars}(q) \to \mathit{types}(t)$, and for each such mapping, we construct a regular expression R_π that describes all traces t' for which there exists a substitution $\mu : \mathit{vars}(q) \cup \Gamma \to \Gamma$ such that μ is an extension of π and $\mu(s) \preccurlyeq_e t'$ for some embedding e that satisfies w and c. Then, we only have to check for each of these mappings π, if the regular expression R_π matches in t. Another approach is to enumerate all embeddings $e : [|s|] \to [|t|]$ that satisfy w and c, and check for each such embedding e whether $\mu(s) \preccurlyeq_e t$ for some substitution μ (which can be done in time $\mathrm{O}(|s|)$, since μ must satisfy $\mu(s) = t[e(1)]t[e(2)] \dots t[e(|s|)]$). From these two algorithms and the obvious dependencies between the parameters, we can directly conclude the statements of the theorem. ◀

Next, we prove the following theorem. Observe that Theorems 4.3 and 4.2 answer Question 4.1 with respect to every subset $p_1, p_2, \dots, p_r$ of the parameters.

▶ **Theorem 4.3.** *Let $k_1, k_2, k_3 \in \mathbb{N}$. $\mathcal{Q}$-Match with $|\mathit{types}(t)| \leqslant k_1$, $\mathit{maxgaps}(c) \leqslant k_2$ and $\mathit{maxc}^+(c) \leqslant k_3$ is NP-complete if $k_1 \geqslant 2$ and $k_2 \geqslant 1$ and $k_3 \geqslant 1$, and it is in P if $k_1 \leqslant 1$ or $k_2 = 0$ or $k_3 = 0$.*

Proof sketch. The theorem's second statement is obtained by simple algorithms. We sketch a hardness reduction that proves the first statement with an example. Let (x_1, x_2, x_4), (x_4, x_5, x_7), and (x_1, x_3, x_5) be the clauses of a Boolean formula in 3-CNF. We construct $s := y_1x_1x_2x_4y_2\ z\ y_3x_4x_5x_7y_4\ z\ y_5x_1x_3x_5y_6$ (where all x_i, y_i and z are variables) and $t :=$ 0001000 1 0001000 1 0001000 (where $\Gamma = \{0, 1\}$). It can be verified that if $\mu(s) \preccurlyeq_e t$ with $\mu(z) = 1$ and e is such that all gaps are 0 or 1, except the gaps between x_i variables, which are 0, then $\mu(x_i) = 1$ for exactly one variable in each clause (i. e., the Boolean formula is "1-in-3 satisfiable"). In order to force $\mu(z) = 1$, we have to add a sufficient number of occurrences of z and 1 as prefixes to s and t, respectively. ◀

Our results discussed so far show that for every parameter $p \in \{|t|, w, |s|, \mathit{repvars}(s)\}$, the problem $\mathcal{Q}$-Match can be solved in polynomial time if p is bounded by a constant. However, the degree of the polynomial may depend on p, which even for small p may lead to prohibitively high running times. Therefore, we ask next for which parameters p (or parameter combinations $(p_1, \dots, p_r)$), we can achieve more attractive running times of the form $f(p) \cdot \mathrm{poly}(|q|, |t|)$, for some computable function f. More precisely, we investigate the parameterised complexity of $\mathcal{Q}$-Match by asking the following questions:

▶ **Question 4.4.** *Is $\mathcal{Q}$-Match parameterised by $(p_1, p_2, \dots, p_r)$ in FPT (i. e., can it be solved in time $\mathrm{O}(f(p_1, p_2, \dots, p_r) \cdot g(|s|, |t|))$ for some polynomial g and a computable function f), or is this not the case (subject to common complexity theoretical assumptions)?*

An answer for parameters $(|\mathit{types}(t)|, \mathit{maxgaps}(c), \mathit{maxc}^+(c))$ follows from Theorem 4.3:

▶ **Corollary 4.5.** *$\mathcal{Q}$-Match parameterised by $(|\mathit{types}(t)|, \mathit{maxgaps}(c), \mathit{maxc}^+(c))$ is not in FPT (unless P = NP).*

From the brute-force approaches and the (more or less) obvious dependencies between the parameters, we can conclude fixed-parameter tractability for several parameterisations:

▶ **Theorem 4.6.** *$\mathcal{Q}$-Match is in FPT with respect to any of the following parameterisations: $(|\mathit{types}(t)|, |\mathit{repvars}(s)|)$; $(|s|, \mathit{maxgaps}(c))$; $(|t|)$; (w); $(|s|, \mathit{maxc}^+(c))$; $(|s|, \mathit{types}(t))$.*

Next, we show that all remaining parameterisations that are not covered by Theorem 4.6 or Corollary 4.5 yield W[1]-hard variants, which are therefore not in FPT (modulo the common complexity theoretical assumption that $\mathsf{W}[1] \neq \mathsf{FPT}$). We assume that the reader is familiar with basic concepts of parameterised complexity theory (cf., e.g., [12, 17, 13]).

▶ **Theorem 4.7.** *Each of the following two problems is* W[1]*-hard:*
- $\mathcal{Q}$-Match *parameterised by* $(|\mathit{repvars}(s)|, \mathit{maxgaps}(c) \leqslant 1, \mathit{maxc}^{+}(c) \leqslant 1)$,
- $\mathcal{Q}$-Match *parameterised by* $(|s|, |\mathit{repvars}(s)|)$.

Proof sketch of Theorem 4.7. We devise suitable reductions from the following problem, which is known to be W[1]-hard with respect to parameter k (cf., [12]):

Multi-Colored Clique (MCClique). The input is a graph $G = (V, E)$ with a partition $V_1, V_2, \ldots, V_k$ of V such that each V_i is an independent set. The question is whether G has a clique of size k (note that such a clique has to contain exactly one vertex from each V_i).

Let $G = (V, E)$ be a graph with a partition $V_1, V_2, \ldots, V_k$ of V such that every V_i is an independent set. Let $n := |V|$, $p_i := |V_i|$ and $V_i = \{v^i_1, v^i_2, \ldots, v^i_{p_i}\}$ for all $i \in [k]$. We have to construct a trace t and an swg-query $q = (s, w, c)$ such that $t \models q$ iff G has a clique of size k. For constructing the trace t, we use a distinct type for every $v \in V$, and we also use two special types \$ and $\diamond$. The query string s will be built using distinct variables $x_1, x_2, \ldots, x_k$, $z_\$$, $y_1, \ldots, y_r$ where r is a suitably chosen number of size polynomial in n and k (the precise choice of r will become clear below, from the construction of s and t). The variables $x_1, x_2, \ldots, x_k$ correspond to the k vertices of the clique, and the variable $z_\$$ will be used as a separator. The variables $x_1, x_2, \ldots, x_k, z_\$$ may be repeated variables of s, while each of the variables $y_1, \ldots, y_r$ will occur in s only once. To avoid notational clutter, we will denote all occurrences of variables y_i simply by y and keep in mind that each occurrence of y is actually an occurrence of an individual variable from $\{y_i \ : \ i \in [r]\}$. Our choice of s and t relies on some gadgets, which we discuss next. The gadget s_V lists all variables x_i separated by $y^n z_\$ y^n$, and t_V lists the vertices of the sets V_i separated by $\diamond^n \$ \diamond^n$. More formally,

$$\begin{array}{llllllllll}
s_V := & y^n & x_1 & y^n\ z_\$\ y^n & x_2 & y^n\ z_\$\ y^n & \ldots & y^n\ z_\$\ y^n & x_k & y^n \\
t_V := & \diamond^n & v^1_1 \ldots v^1_{p_1} & \diamond^n\ \$\ \diamond^n & v^2_1 \ldots v^2_{p_2} & \diamond^n\ \$\ \diamond^n & \ldots & \diamond^n\ \$\ \diamond^n & v^k_1 \ldots v^k_{p_k} & \diamond^n
\end{array}$$

The purpose of this gadget is as follows: We can show that if $\mu(s_V) \preccurlyeq t_V$ for some $\mu : \mathsf{Vars} \to \Gamma$ with $\mu(z_\$) = \$$, then $(\mu(x_1), \mu(x_2), \ldots, \mu(x_k)) \in V_1 \times V_2 \times \cdots \times V_k$, and there exists an embedding e with *maximum gap size* 1 (i.e., $\max\{e(i{+}1){-}1{-}e(i) \colon i \in [|s_V|{-}1]\} \leqslant 1$) and a substitution μ' that differs from μ only on variables in $\{y_1, \ldots, y_r\}$, such that $\mu'(s_V) \preccurlyeq_e t_V$. Note that for ensuring the latter property, we need the occurrences of y^n.

In order to enforce that the set $\{\mu(x_1), \mu(x_2), \ldots, \mu(x_k)\}$ is a clique, we use the following gadgets. For every i, j with $1 \leqslant i < j \leqslant k$, let $(u^i_1, u^j_1), \ldots, (u^i_{q_{i,j}}, u^j_{q_{i,j}})$ be a list of exactly the edges between V_i and V_j, and let this list be ordered lexicographically. We let

$$s_{i,j} := \quad y^{3n^2}\ (x_i x_j)^3\ y^{3n^2} \quad \text{and} \quad t_{i,j} := \quad \diamond^{3n^2} (u^i_1 u^j_1)^3 (u^i_2 u^j_2)^3 \ldots (u^i_{q_{i,j}} u^j_{q_{i,j}})^3\ \diamond^{3n^2}\ .$$

The purpose of this gadget is as follows: We can show that if there is a substitution μ with $\mu(x_i) \in V_i$, $\mu(x_j) \in V_j$ and $\mu(s_{i,j}) \preccurlyeq t_{i,j}$, then $(\mu(x_i), \mu(x_j)) \in E$ and there exists an embedding e with maximum gap size 1 and a substitution μ' that differs from μ only on variables in $\{y_1, \ldots, y_r\}$ such that $\mu'(s_{i,j}) \preccurlyeq_e t_{i,j}$. In particular, this gadget enforces that, for every $1 \leqslant i < j \leqslant k$, μ maps x_i and x_j to *adjacent* vertices from V_i and V_j (for this, we essentially use that the list of edges in $t_{i,j}$ is ordered lexicographically).

Finally, we combine all the gadgets into the query string s and trace t as follows:

$$\begin{array}{lll}
s & := & (z_\$)^{3n^2+1} s_V\ z_\$\ s_{1,2}\ z_\$\ s_{1,3}\ z_\$\ \cdots\ z_\$\ s_{1,k}\ z_\$\ s_{2,3}\ z_\$\ \cdots\ z_\$\ s_{2,k}\ z_\$\ \cdots\ z_\$\ s_{k-1,k} \\
t & := & (\$)^{3n^2+1}\ t_V\ \$\ t_{1,2}\ \$\ t_{1,3}\ \$\ \cdots\ \$\ t_{1,k}\ \$\ t_{2,3}\ \$\ \cdots\ \$\ t_{2,k}\ \$\ \cdots\ \$\ t_{k-1,k}
\end{array}$$

Table 1 An integer entry means that the parameter is bounded by this constant, the entry "**p**" means that the problem is parameterised by this parameter, and the entry "–" means that the parameter is not bounded by a constant and the problem is not parameterised by this parameter.

$\lvert t\rvert$	w	$\lvert s\rvert$	$\lvert \mathit{types}(t)\rvert$	$\lvert \mathit{repvars}(s)\rvert$	$\mathit{maxgaps}(c)$	$\mathit{maxc}^+(c)$	Complexity
p	–	–	–	–	–	–	FPT (Thm. 4.6)
–	**p**	–	–	–	–	–	FPT (Thm. 4.6)
–	–	**p**	–	–	–	**p**	FPT (Thm. 4.6)
–	–	**p**	**p**	–	–	–	FPT (Thm. 4.6)
–	–	**p**	–	**p**	–	–	W[1]-hard (Thm. 4.7)
–	–	**p**	–	–	**p**	–	FPT (Thm. 4.6)
–	–	–	**p**	**p**	–	–	FPT (Thm. 4.6)
–	–	–	2	–	1	1	NP-hard (Thm. 4.3)
–	–	–	–	**p**	1	1	W[1]-hard (Thm. 4.7)

The construction of the query $q = (s, w, c)$ is completed by choosing $w = \infty$ (or $w = |t|$, which does not make any difference here), and $(c_i^-, c_i^+) = (0, 0)$ for every $1 \leqslant i \leqslant 3n^2$, and $(c_i^-, c_i^+) = (0, 1)$ for every $3n^2{+}1 \leqslant i \leqslant |s|{-}1$.

It can now be verified that $t \models q$ if and only if G contains a clique of size k. This completes the proof sketch of the theorem's first statement.

For proving the second statement, we modify the reduction such that we use constant length separators y and $\diamond$ instead of y^n, $\diamond^n$, y^{3n^2} and $\diamond^{3n^2}$. This modification allows us to use prefixes $(z_\$)^2$ and $(\$)^2$ instead of $(z_\$)^{3n^2+1}$ and $(\$)^{3n^2+1}$, which means that the total length of $|s|$ only depends on k (but now neither $\mathit{maxgaps}(c)$ nor $\mathit{maxc}^+(s)$ are bounded anymore). This yields the theorem's second statement. ◀

We note that the theorems from above answer Question 4.4 for every subset $p_1, p_2, \ldots, p_r$ of our parameters – this can be verified with the help of Table 1.

We conclude this section by discussing some related questions. Query strings can also be considered as sequences of distinct variables $x_1 x_2 \cdots x_n$ enriched with constraints "$x_i = x_j$" and "$x_i = \gamma$" for $\gamma \in \Gamma$. If the first type of constraint is not used, then $|\mathit{repvars}(s)| = 0$, and therefore the matching problem is in P (Theorem 4.2). If, instead, we disallow constraints "$x_i = \gamma$" (i. e., query strings must not contain types), then this does not make the matching problem any easier: all our reductions produce query strings that consist of variables only. Giving up the global window size is equivalent to setting it to ∞, which means that the lower bounds of Theorems 4.3 and 4.6 still apply (note that the reduction sets the global window size to ∞). On the other hand, for swg-queries without local gap-size constraints (i. e., with local gap-size constraints of the form $((0, \infty), (0, \infty), \ldots, (0, \infty)))$ the complexity of the matching problem is open. We plan to address this in the full version of this paper.

In order to use the algorithm of Section 3, we need to fix the length ℓ of the query string, the global window size w, and the tuple c of local gap-size constraints. Finding suitable values for ℓ and c seems comparatively simple: firstly, we can afford to try out several combinations (e. g., all combinations of values $\ell \in \{5, 6, \ldots, 10\}$ and $w \in \{100, 1000, 5000\}$, which requires 18 runs of the algorithm), and, secondly, it seems likely that some a-priori knowledge of the data will lead to reasonable choices for these parameters. The tuple of local gap-size constraints, on the other hand, is a more complex parameter and, especially if we want to try the algorithm on different query string sizes, it seems likely that the best we can do is to give a reasonable gap-size constraint for *all* gaps, i. e., local gap-size constraints $((i, j), (i, j), \ldots, (i, j))$. Whether the matching problem for such swg-queries is also NP-hard is not answered by the reduction of Theorem 4.3, since it uses different types of gap-size constraints. However, the reduction can be easily adapted: Instead of the factors $y_1x_1x_2x_4y_2$ of the query string, we use factors $y_1y_2y_3(x_1)^2(x_2)^2(x_4)^2y_4y_5y_6$, and instead of the factors 0001000 of the trace, we use factors 0^7110^7. It can be verified that if $\mu(s) \preccurlyeq_e t$ with $\mu(z) = 1$ and e is such that *all gaps* are 0 or 1, then $\mu(x_i) = 1$ for exactly one variable in each clause.

5 Practical Considerations and Concluding Remarks

Motivated by the task of discovering patterns of interest in event streams, we introduced swg-queries (Section 2), we formalised query discovery as computing descriptive queries (Section 3), we presented an algorithm for this task (Algorithm 1), and we provided an in-depth complexity analysis of the problem of query discovery (Section 4).

What are possible practical implications of our theoretical study? To approach this question, let us briefly discuss how the complexity bounds of Section 4 can help to manage our expectations. Let $\|(\ell, w, c)\|$ and $\|\mathcal{S}\|$ be the size of reasonable encodings of (ℓ, w, c) and $\mathcal{S}$, respectively. In a practical scenario, we can expect $\|\mathcal{S}\|$ (and therefore possibly also $\|w\|$ and $\|c\|$, see Remark 3.3) to be large, but we can assume a moderate query string length ℓ, since queries should still be understandable by users. Hence, running times $\mathrm{O}(f(\ell)g(\|\mathcal{S}\|, \|(\ell, w, c)\|))$ might be practically relevant even for exponential function f and low-degree polynomial g. Unfortunately, Theorem 4.7 and the reduction of Theorem 3.6 rule out such algorithms. On the other hand, Theorems 4.6 and 3.5 show that we can indeed solve $\mathcal{Q}^{|rv(s)|\leqslant k}$-$\mathsf{CompDescQuery}$ in time $\mathrm{O}(f(k, |\mathit{types}(\mathcal{S})|)g(\|\mathcal{S}\|, \|(\ell, w, c)\|))$ for polynomial g, where $\mathcal{Q}^{|rv(s)|\leqslant k}$ is the class of queries with $|\mathit{repvars}(q)| \leqslant k$. This might be of practical interest, since our examples discussed in Section 1 suggest that $|\mathit{repvars}(q)|$ and $|\mathit{types}(\mathcal{S})|$ are small. In fact, even a restriction like, say $|\mathit{repvars}(q)| \leqslant 7$ and $|\mathit{types}(\mathcal{S})| \leqslant 15$, seems still practically relevant, and in this case our algorithm has a low-degree polynomial running time.

These theoretical considerations justify hope that, despite the inherent hardness of query discovery, our algorithm is also practically worthwhile. As a proof of concept and in order to investigate our algorithms' performance on real world data, we implemented a prototype in Python and conducted experiments on Google Cluster Traces [26]. We summarise our experiments' main results as follows:

- We discovered queries from the Google Cluster Traces, including queries that contain situations of interest. However, we also discovered queries that do not contain real situations of interest (but still are descriptive according to Definition 3.1).
- Considering different query string lengths or support thresholds, we observed expected correlations concerning the overall runtime and the number of queries q' for which the support $\mathsf{supp}(q', \mathcal{S})$ is computed.
- Comparing datasets regarding runtime (or the number of queries q' for which the support $\mathsf{supp}(q', \mathcal{S})$ is computed), for a fixed query string length, it turns out that they do not only depend on the dataset size, but also on the order in which the positions of the query string are considered, and on whether variable operations have to be tested or not.

As future work, we plan to further develop our prototype implementation and extend our initial experiments to a comprehensive experimental analysis. In particular, we are interested in evaluating heuristics and to explore possible ways for improving the algorithm's practical performance. For example, treating the check of $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ as $|\mathcal{S}|$ independent instances of the matching problem makes sense for our theoretical considerations, but in practice it might be possible to exploit the fact that q' does not change for these $|\mathcal{S}|$ instances, or that in each run of the algorithm $\mathcal{S}$ is the same for *all* checks of $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$, or that whenever we check $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$, we already checked $\mathsf{supp}(q'', \mathcal{S}) \geqslant \mathsf{sp}$ for a rather similar query q''.

References

1 Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994. URL: http://www.vldb.org/conf/1994/P487.PDF.

2 Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee L. P. Chen, editors, *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 3–14. IEEE Computer Society, 1995. doi:10.1109/ICDE.1995.380415.

3 Dana Angluin. Inductive inference of formal languages from positive data. *Inf. Control.*, 45(2):117–135, 1980. doi:10.1016/S0019-9958(80)90285-5.

4 Alexander Artikis, Chris Baber, Pedro Bizarro, Carlos Canudas-de-Wit, Opher Etzion, Fabiana Fournier, Paul Goulart, Andrew Howes, John Lygeros, Georgios Paliouras, Assaf Schuster, and Izchak Sharfman. Scalable proactive event-driven decision making. *IEEE Technol. Soc. Mag.*, 33(3):35–41, 2014. doi:10.1109/MTS.2014.2345131.

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. ACM, 2017. doi:10.1145/3093742.3095106.

6 Alexander Artikis, Matthias Weidlich, François Schnitzler, Ioannis Boutsis, Thomas Liebig, Nico Piatkowski, Christian Bockermann, Katharina Morik, Vana Kalogeraki, Jakub Marecek, Avigdor Gal, Shie Mannor, Dimitrios Gunopulos, and Dermot Kinane. Heterogeneous stream processing and crowdsourcing for urban traffic management. In Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos, and Vincent Leroy, editors, *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*, pages 712–723. OpenProceedings.org, 2014. doi:10.5441/002/edbt.2014.77.

7 Lars Baumgärtner, Christian Strack, Bastian Hoßbach, Marc Seidemann, Bernhard Seeger, and Bernd Freisleben. Complex event processing for reactive security monitoring in virtualized computer systems. In Frank Eliassen and Roman Vitenberg, editors, *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15, Oslo, Norway, June 29 - July 3, 2015*, pages 22–33. ACM, 2015. doi:10.1145/2675743.2771829.

8 Lei Chang, Tengjiao Wang, Dongqing Yang, and Hua Luan. Seqstream: Mining closed sequential patterns over stream sliding windows. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 83–92. IEEE Computer Society, 2008. doi:10.1109/ICDM.2008.36.

9 Hong Cheng, Xifeng Yan, and Jiawei Han. Incspan: incremental mining of sequential patterns in large database. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 527–532. ACM, 2004. doi:10.1145/1014052.1014114.

10 Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012. doi:10.1145/2187671.2187677.

11 Joel D. Day, Pamela Fleischmann, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. The edit distance to k-subsequence universality. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, pages 25:1–25:19, 2021. doi:10.4230/LIPIcs.STACS.2021.25.

12 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

13 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

14 Henning Fernau, Florin Manea, Robert Mercas, and Markus L. Schmid. Revisiting Shinohara's algorithm for computing descriptive patterns. *Theor. Comput. Sci.*, 733:44–54, 2018. doi:10.1016/j.tcs.2018.04.035.

15 Henning Fernau and Markus L. Schmid. Pattern matching with variables: A multivariate complexity analysis. *Inf. Comput.*, 242:287–305, 2015. doi:10.1016/j.ic.2015.03.006.

16 Henning Fernau, Markus L. Schmid, and Yngve Villanger. On the parameterised complexity of string morphism problems. *Theory Comput. Syst.*, 59(1):24–51, 2016. doi:10.1007/s00224-015-9635-3.

17 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

18 Dominik D. Freydenberger and Daniel Reidenbach. Existence and nonexistence of descriptive patterns. *Theor. Comput. Sci.*, 411(34-36):3274–3286, 2010. doi:10.1016/j.tcs.2010.05.033.

19 Dominik D. Freydenberger and Daniel Reidenbach. Inferring descriptive generalisations of formal languages. *J. Comput. Syst. Sci.*, 79(5):622–639, 2013. doi:10.1016/j.jcss.2012.10.001.

20 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.

21 Lars George, Bruno Cadonna, and Matthias Weidlich. Il-miner: Instance-level discovery of complex event patterns. *Proc. VLDB Endow.*, 10(1):25–36, 2016. doi:10.14778/3015270.3015273.

22 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.

23 Florin Manea and Markus L. Schmid. Matching patterns with variables. In *Combinatorics on Words - 12th International Conference, WORDS 2019, Loughborough, UK, September 9-13, 2019, Proceedings*, pages 1–27, 2019. doi:10.1007/978-3-030-28796-2_1.

24 Alessandro Margara, Gianpaolo Cugola, and Giordano Tamburrelli. Learning from the past: automated rule generation for complex event processing. In Umesh Bellur and Ravi Kothari, editors, *The 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, May 26-29, 2014*, pages 47–58. ACM, 2014. doi:10.1145/2611286.2611289.

25 A. Mateescu and A. Salomaa. Patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 230–242. Springer, 1997.

26 Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster-usage traces: format+schema. *Google Inc., White Paper*, pages 1–14, 2011.

27 T. Shinohara. Polynomial time inference of pattern languages and its application. In *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science, MFCS*, pages 191–209, 1982.

28 Takeshi Shinohara and Setsuo Arikawa. Pattern inference. In *Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report*, pages 259–291, 1995. doi:10.1007/3-540-60217-8_13.

29 Kia Teymourian, Malte Rohde, and Adrian Paschke. Knowledge-based processing of complex stock market events. In Elke A. Rundensteiner, Volker Markl, Ioana Manolescu, Sihem Amer-Yahia, Felix Naumann, and Ismail Ari, editors, *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*, pages 594–597. ACM, 2012. doi:10.1145/2247596.2247674.

30 Jianyong Wang and Jiawei Han. BIDE: efficient mining of frequent closed sequences. In Z. Meral Özsoyoglu and Stanley B. Zdonik, editors, *Proceedings of the 20th International*

Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA, pages 79–90. IEEE Computer Society, 2004. `doi:10.1109/ICDE.2004.1319986`.

31 Haopeng Zhang, Yanlei Diao, and Neil Immerman. On complexity and optimization of expensive queries in complex event processing. In Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 217–228. ACM, 2014. `doi:10.1145/2588555.2593671`.