Reaching New Limits: Discovery of Multi-Dimensional Disjunctive Subsequence-Queries with Intervals

André Frochaux [1], Sarah Kleest-Meißner [1], and Benjamin Scheidt [1]

Abstract: A query model for sequence data was introduced in Kleest-Meißner et al. (2022) in the form of subsequence-queries with wildcards and gap-size constraints (swg-queries, for short). These queries consist of a pattern over an alphabet of variables and values, as well as a global window size and a number of local gap-size constraints.

Based on previous extensions of swg-queries, namely multi-dimensional swg-queries and disjunctive swg-queries, we converge to common languages in the field of Complex Event Processing by introducing multi-dimensional disjunctive subsequence-queries with intervals. This pushes the expressive power of the query language to certain kinds of inequalities as well. We discuss a suitable characterisation of containment, a classical property considered in database theory, and adapt results concerning the discovery of (multi-dimensional or disjunctive) swg-queries to multi-dimensional disjunctive subsequence-queries with intervals.

Keywords: Subsequence-queries, disjunction, inequalities, interval, learning descriptive queries, subsequences, embeddings.

1 Introduction

Within the field of complex event processing (CEP) systems continuously evaluate queries over sequence data, e. g. a stream of events, to recognize specific event patterns [4, 11]. These event queries specify event patterns that are materialisations of a situation of interest, e. g. abnormal job execution or fraudulent transaction. Their recognition enables reactive and proactive applications in various domains, e. g. cluster monitoring scenarios [19], urban transportation [3], and computational finance [18]. Query languages for CEP usually allow for correlation of events, constraints on the ordering of events (sequencing) and their occurrence within a certain window over the sequence data. Further conditions over the attribute values of events may be defined using operators like conjunction, disjunction, Kleene closure, negation and variables which may be bound to events in the sequence data [2, 11, 20].

The *finite boolean query evaluation problem for CEP* receives as input an event query q and a finite sequence of events t (called trace) and outputs "yes" if, and only if, the trace matches the query.

[1] Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany,
andre.frochaux@informatik.hu-berlin.de, https://orcid.org/0009-0001-7918-8725;
kleemeis@informatik.hu-berlin.de, https://orcid.org/0000-0002-4133-7975;
benjamin.scheidt@hu-berlin.de, https://orcid.org/0000-0003-2379-3675

Example 1. Consider a cluster monitoring scenario in which each of the following traces describes the lifecycle transitions of a job with abnormal execution:

t_1	$\coloneqq$	$\langle$SCH, 1$\rangle$	$\langle$FAI, 1$\rangle$	$\langle$SCH, 9$\rangle$	$\langle$UPD, 9$\rangle$	$\langle$FIN, 1$\rangle$
t_2	$\coloneqq$	$\langle$UPD, 1$\rangle$	$\langle$FAI, 1$\rangle$	$\langle$UPD, 9$\rangle$	$\langle$FIN, 1$\rangle$	
t_3	$\coloneqq$	$\langle$SUB, 1$\rangle$	$\langle$FAI, 1$\rangle$	$\langle$SUB, 9$\rangle$	$\langle$SCH, 8$\rangle$	$\langle$FIN, 1$\rangle$

Each event captures a task during the job execution, i. e. the kind of the task, e. g. SCH, and the ID of the priority of the task, e.g. 5. The abbreviations SUB, SCH, UPD, FAI, or FIN stand for the tasks: submit, schedule, update, fail, or finish for the corresponding job.

In the following, we depict a query q that describes the abnormal execution of a job that has a reoccurring task (with different priorities) due to the failed task in between the reoccurring event, before it gets finished. The syntax of the query follows common languages for CEP [11]:

$$
\begin{array}{lll}
q = & \text{PATTERN SEQ} & (e_1, e_2, e_3, e_4) \\
& \text{WHERE} & e_1.\text{task} = e_3.\text{task AND} \\
& & e_2.\text{task} = \text{FAI AND } e_4.\text{task} = \text{FIN AND} \\
& & e_1.\text{prio} = e_2.\text{prio} = e_4.\text{prio} = 1 \text{ AND} \\
& & e_3.\text{prio} = 9 \\
& \text{WITHIN} & 5 \text{ events}
\end{array}
$$

Note that q is matched by each of the three traces t_1, t_2, t_3, illustrated for q and t_1 as follows:

$$
\begin{array}{lccccccc}
 & & e_1 & & e_2 & & e_3 & & & e_4 \\
t_1 = & \langle \text{SCH}, 1 \rangle & & \langle \text{FAI}, 1 \rangle & & \langle \text{SCH}, 9 \rangle & & \langle \text{UPD}, 9 \rangle & & \langle \text{FIN}, 1 \rangle.
\end{array}
$$

⌟

While such queries are a comprehensive characterisation of situations of interest, finding a suitable query is challenging and often assumed to be a manual step. Previous work therefore proposed to automatically discover event queries from a sample of historic traces, where each trace contains the situation of interest [10, 15]. Discovered queries should be *correct* in the sense that they match a sufficiently large subset of traces (w.r.t. to a *support threshold* sp) and *descriptive*, i. e. each query that is more specific is no longer a correct query.

Recently, Kleest-Meißner et al. [12, 13] and Frochaux et al. [8] investigated the problem of discovering event queries from a finite sample of traces from a theoretical point of view. They proposed *multi-dimensional subsequence queries with wildcards and gap-size constraints* (*mswg-queries*) as well as different extensions (disjunctive queries and queries with generalised gap constraints), as a formal model that covers the essence of the task of event query discovery. Let k denote the number of attributes per event. In essence, an mswg-query is a string s consisting of k-tuples over an alphabet of variables and values, a global window size w, and a tuple of local gap-size constraints c.

Example 2. The query from Example 1 can be translated into the mswg-query $q = (s, w, c)$ with

$$s \quad = \quad \langle x, 1\rangle \, \langle \mathsf{FAI}, 1\rangle \, \langle x, 9\rangle \, \langle \mathsf{FIN}, 1\rangle,$$

for variable x and values FAI, FIN, 1, and 9, $w = 5$, and $c = ((0, \infty), (0, 0), (0, 1))$. Note that c can not be directly derived from the query in Example 1 but is defined in a way that the traces t_1 to t_3 match the query (any tuple of the form $c = ((0, i), (0, j), (0, k))$, with $2 \leqslant i, j \in \mathbb{N} \cup \{\infty\}$ and $k \in \mathbb{N}_{\geqslant 1} \cup \{\infty\}$ would be a suitable choice). We illustrate a witness replacement of variables for q and t_1 as follows:

$$\begin{array}{lcccccc} s & = & \langle x, 1\rangle & \langle \mathsf{FAI}, 1\rangle & \langle x, 9\rangle & & \langle \mathsf{FIN}, 1\rangle \\ t & = & \langle \mathsf{SCH}, 1\rangle & \langle \mathsf{FAI}, 1\rangle & \langle \mathsf{SCH}, 9\rangle & \langle \mathsf{UPD}, 9\rangle & \langle \mathsf{FIN}, 1\rangle \end{array}$$ ⌟

The query model allows for sequencing, variables that range over single event attributes, (disjunctions for the corresponding extension introduced in [8]) and different kinds of window size constraints. Hence, mswg-queries combine core features of complex event processing languages with classical concepts of theoretical computer science: subsequences and (string) patterns with variables.

Patterns with variables were introduced by Angluin [1] and build the core of inductive inference, formal language theory and combinatorics on words [17, 14, 16]. Syntactically, mswg-queries are Angluin-style patterns enriched by different kinds of window sizes, and with a semantics that is adapted to event based scenarios: here, variables refer solely to single values whereas in Angluin's semantics they refer to finite sequences of values.

For mswg-queries, an algorithm was provided that, on input of a sample $\mathcal{S}$, a support threshold sp, and further parameters on the query, outputs a query q that is correct and descriptive for $\mathcal{S}$, sp, and the parameters on the query. The algorithmic idea stems from the field of Angluin-style Pattern Languages, for which Shinohara's algorithm computes a descriptive pattern upon input of a sample $\mathcal{S}$ and a support threshold $\mathsf{sp} = 1.0$ (see also [6] for an analysis and extensions of Shinohara's algorithm).

However, mswg-queries and their extensions only support filtering based on equality constraints, equivalent to the constraints depicted in Example 1, while systems like the IL-Miner [10] also allow for the discovery of queries that support lower and upper bounds on variables, which can be modelled as intervals.

Contribution 1. In this paper, we address this limitation by merging multi-dimensional swg-queries and disjunctive swg-queries and extending the expressive power of the resulting query model even further to *multi-dimensional disjunctive subsequence-queries with intervals* (*mdswg$^{\leqslant}$-queries*, for short, where $\leqslant$ hints at the order domains may have). Different from previous work on swg-queries, we also extend the underlying data model slightly by considering an *event schema*, i. e., a tuple of attributes where each attribute may have its own domain.

Example 3. Let us consider the same scenario as in Example 1 but with a slightly modified set of traces over the schema (task, priority). Let

t_1	$\coloneqq$	$\langle$SCH, 1$\rangle$	$\langle$FAI, 1$\rangle$	$\langle$SCH, 9$\rangle$	$\langle$UPD, 9$\rangle$	$\langle$FIN, 1$\rangle$,
t_2	$\coloneqq$	$\langle$UPD, 2$\rangle$	$\langle$FAI, 1$\rangle$	$\langle$UPD, 10$\rangle$	$\langle$FIN, 1$\rangle$,	
t_3	$\coloneqq$	$\langle$SUB, 1$\rangle$	$\langle$KIL, 1$\rangle$	$\langle$SUB, 9$\rangle$	$\langle$SCH, 8$\rangle$	$\langle$FIN, 1$\rangle$.

Note that the priorities do not carry the same values 1 or 9 as in Example 1, and that the faulty behaviour between the reoccurring task now is either FAI *or* KIL. This can be captured in our query model by allowing for the disjunction of the values FAI and KIL and intervals describing the priorities as follows:

$$q = (\langle x, [1,2]\rangle \langle(\mathsf{FAI}|\mathsf{KIL}), 1\rangle \langle x, [9,10]\rangle \langle \mathsf{FIN}, 1\rangle, w, c) \ . \qquad \lrcorner$$

Note that we are not restricted to integers for ordered domains, but allow for totally ordered sets in general. One might ask whether intervals can be modelled as disjunctions, since each interval could be modelled as a disjunction of its values. Note that this becomes difficult if we consider for example $\mathbb{R}$ as a domain. Furthermore, as described in [8], disjunctions may lead to an exponential blow-up in the number of tests whether a current query is matched by the traces used for query discovery. It will turn out, that this is not the case for intervals which can be computed with only a logarithmic number of tests.

We would like to stress the point that an interval can be interpreted as lower and upper bounds on values (and vice versa) that may occur in a trace, as depicted in the next example. On the other hand, on a finite sample, defining *inclusive* lower *and* upper bounds, i. e., an interval, is *always* more restrictive than finding only a (non-inclusive) lower *or* upper bound. Hence, we only consider intervals in this paper.

Example 4. Suppose we want to detect a fire based on a given stream of events measuring temperature and humidity. The following query captures a possible pattern for fire detection, comprising a measured temperature higher than or equal to $45°C$ followed by a humidity measurement of less than or equal to 25%:

$$s = (\geqslant 45°C, x_h)\ (x_t, \leqslant 0.25)$$

Note that a temperature measurement of $\geqslant 45°C$ can also be seen as an interval $[45°C, x]$ for a reasonable upper bound $x > 45°C$. $\lrcorner$

Contribution 2. We discuss a suitable characterisation of containment, and, based thereon adapt results concerning the discovery of multi-dimensional or disjunctive swg-queries to mdswg$^{\leqslant}$-queries. Note that this extension of the query model leads to a more intuitive way of defining queries. We believe that queries with intervals are less prone to overfitting compared to queries that only support disjunctions.

Organisation. The rest of this paper is organised as follows. We introduce some basic notation in Section 2. Section 3 is devoted to the definition of mdswg$^{\leqslant}$-queries and the discussion of a suitable characterisation of query containment. We define the query discovery problem and provide an algorithm to solve this problem in Section 4, before we summarise our results and indicate directions of future work in Section 5.

2 Preliminaries

We assume that the reader is familiar with common mathematical concepts and notations. Let $\mathbb{N}$ and $\mathbb{N}_{\geqslant 1}$ be the set of non-negative integers and positive integers, respectively. For $n \in \mathbb{N}$, we let $[n] \coloneqq \{\, i \in \mathbb{N}_{\geqslant 1} \,:\, 1 \leqslant i \leqslant n \,\}$. The symbols $\subseteq$, $\subsetneq$, $\supseteq$ and $\supsetneq$ denote the subset, proper subset, superset and proper superset relation, respectively. We call a set (and thus also in particular: an interval) *singleton*, if it contains precisely one value.

Let A be a non-empty set. We write A^* (and A^+) for the set of all strings (and the set of all non-empty strings) over A. We denote the length ℓ of a string $s = a_1 \dots a_\ell \in A^*$ by $|s|$. For a position $i \in [\ell]$, we write $s[i]$ to denote the letter a_i at position i in s. A *factor* of a string $s \in A^*$ is a string $v \in A^*$ such that $s = uvu'$ for some $u, u' \in A^*$. If $|u| = 0$ ($|u'| = 0$), the factor v is also called a *prefix (suffix) of* s. We write $s[i, j]$ for positions $1 \leqslant i \leqslant j \leqslant \ell$ for the factor $a_i \dots a_j$ of s starting at position i and ending at position j, i. e., including i and j.

A *subsequence* of a string $t = t_1 t_2 \dots t_n \in A^*$ is a string $s = s_1 \dots s_m$ where $m \leqslant n$ and there exist integers $1 \leqslant i_1 < \dots < i_m \leqslant n$ such that $s_j = t_{i_j}$ for all $j \in [m]$; the mapping $\xi \colon [m] \to [n]$ with $\xi(j) = i_j$ for all $j \in [m]$ is called an *embedding* of s in t.

Example 5. The string $s = \mathsf{ACC}$ is a subsequence of the string $t = \mathsf{ABACCB}$ with embedding ξ where $\xi(1) = 3$, $\xi(2) = 4$ and $\xi(3) = 5$. Note that there exists a second embedding for s in t, namely ξ' with $\xi'(1) = 1$, $\xi'(2) = 4$ and $\xi'(3) = 5$. ⌟

For a comprehensive list of relevant work and deeper insights into subsequences or formal language theory, we refer the reader to the introductions of recent papers [5, 9].

Definition 6 (Schema)**.** An *event schema* of *dimension* $k \in \mathbb{N}_{\geqslant 1}$ (*schema*, for short) is a tuple $\mathcal{A} = (A_1, \dots, A_k)$ of attributes A_i. Each attribute A_i is defined over a (possibly infinite) domain denoted by $dom(A_i)$ and a domain *may be* totally ordered via $\leqslant_{A_i}$. The set of all attribute values is $\Gamma \coloneqq \bigcup_{i=1}^{n} dom(A_i)$. ⌟

Note that the different domains do not have to be disjoint. However, we do assume that the ordered domains are disjoint from the unordered domains. Without loss of generality, one may assume that there exists a single event schema, since different event schemas could be summarised by a single schema which is defined by the disjoint union of multiple schemas.

Definition 7 (Events and Traces)**.** Let $k \in \mathbb{N}_{\geqslant 1}$ and let $\mathcal{A}$ be a schema of dimension k. An *event* $e = \langle a_1, \dots, a_k \rangle$ (over $\mathcal{A}$) is a tuple of attribute values that instantiates the event schema $\mathcal{A}$, i. e., $a_j \in dom(A_j)$ for all $j \in [k]$. We write $e[j]$ to refer to the j-th value of event e, i. e., the attribute value a_j.

A *k-dimensional event trace over* $\mathcal{A}$ (or *trace* for short) is a finite sequence of events $t = e_1, \ldots, e_n$ over $\mathcal{A}$. The length $|t|$ of t is the number of events it comprises, i. e., $|t| = n$. We write $t[i]$ ($t[i][j]$) to denote the (j-th attribute of the) i-th event in t, for $i \in [n]$, $j \in [k]$. The set of attribute values occurring in a trace t, also called the *active domain* of t, is defined as $adom(t) = \{\gamma \in \Gamma : \text{there ex. } i \in [n],\ j \in [k] \text{ s.t. } t[i][j] = \gamma\}$. ⌟

Example 8. Consider again 2-dimensional events with attributes task and priority, i. e., $\mathcal{A}_{\text{ex}} \coloneqq (\mathsf{task}, \mathsf{priority})$ with $dom(\mathsf{task}) = \{\mathsf{SUB}, \mathsf{SCH}, \mathsf{UPD}, \mathsf{KIL}, \mathsf{FAI}, \mathsf{FIN}\}$ and $dom(\mathsf{priority}) = \{1, \ldots, 10\}$, where $dom(\mathsf{priority})$ is ordered (by the natural order) and $dom(\mathsf{task})$ is unordered. The traces in Example 3 are traces over $\mathcal{A}_{\text{ex}}$, consisting of a finite number of events instantiating the given schema. They are of length $|t_1| = |t_3| = 5$ and $|t_2| = 4$. ⌟

For the remainder of this paper, we fix a dimension k and an event schema $\mathcal{A} = (A_1, \ldots, A_k)$. We require that $|dom(A_i)| \geqslant 2$ for all $i \in [k]$ — otherwise, we could omit the whole attribute, since it would not contain any information.

3 Query Model

A query describes a sequence of events, that should occur in a trace, in form of a query string. The occurrence of these events in a trace should thereby satisfy different kinds of window constraints. Each position $s[i][j]$ in the query string may either carry a variable, a single value from $dom(A_j)$, an interval which is a finite subset of $dom(A_j)$ (if $dom(A_j)$ is totally ordered), or a set of values $\chi \subseteq_{fin} dom(A_j)$ (if $dom(A_j)$ is not totally ordered). The latter denotes the disjunction of the values in χ and is called a *disjunctive clause*. To keep the definitions simple, we model single values via singleton sets or singleton intervals. Thus, a query is formalised as follows.

Let $\mathbb{V}$ be an infinite set of variables with $\mathbb{V} \cap \Gamma = \emptyset$, let O be the set of indices of the ordered, and U the set of indices of the unordered attributes of $\mathcal{A}$. For all $i \in O$, let $\mathbb{I}_i$ be the set of all closed intervals over $dom(A_i)$ and for all $j \in U$, let $\mathbb{D}_j$ be the set of all finite non-empty subsets over $dom(A_j)$. Further, let $\mathbb{I} \coloneqq \bigcup_{i \in O} \mathbb{I}_i$ and $\mathbb{D} \coloneqq \bigcup_{j \in U} \mathbb{D}_j$.

Definition 9. A k-dswg$^{\leqslant}$-query $q = (s, w, c)$ (over $\mathcal{A}$) is specified by

- a query string $s = s[1] \ldots s[\ell]$, where each $s[i]$ is a k-tuple and f.a. $(i, j) \in [\ell] \times [k]$
$$s[i][j] = \begin{cases} x \in \mathbb{V}, & \text{or} \\ [p, q] \in \mathbb{I}_j, & \text{if } j \in O, \text{ or} \\ \chi \in \mathbb{D}_j, & \text{if } j \in U; \end{cases}$$
- a global window size $w \in \mathbb{N}_{\geqslant 1} \cup \{\infty\}$ with $w \geqslant \ell$; and
- a tuple $c = (c_1, c_2, \ldots, c_{\ell-1})$ of local gap-size constraints (for ℓ 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 $i \in [\ell{-}1]$ and $\ell + \sum_{i=1}^{\ell-1} c_i^- \leqslant w$.

As for traces, the length of the query string equals the number of k-tuples it comprises. ⌟

Example 10. Consider the schema $\mathcal{A}_{\text{ex}}$ from Example 8 and let

$$s \coloneqq \langle x, [1,2]\rangle \langle \{\mathsf{FAI}, \mathsf{KIL}\}, [1,1]\rangle \langle x, [9,10]\rangle \langle \{\mathsf{FIN}\}, [1,1]\rangle,$$
$$w \coloneqq 5, \text{ and } c \coloneqq \{ (0,0), (0,0), (0,1) \}$$

be the query string, global window size and local gap-size constraints from Example 3. The tuple $q \coloneqq (s, w, c)$ is an example of a 2-dswg$^{\leqslant}$-query. Intuitively, it expresses the abnormal execution of a job: a task is performed twice with increasing priority due to a kill or failure events in between, before the job is finally finished. ⌟

For the sake of readability, we omit braces in query strings if it is one dimensional, or if a disjunctive clause or an interval consists only of a unique element. Further we may write disjunctive clauses in the usual way, rather than as sets. For example, we may write the query string $\langle \mathsf{A}, x_1\rangle \langle x_2, 4\rangle \langle (\mathsf{A} \mid \mathsf{C}), [2,7]\rangle$ instead of $\langle \{\mathsf{A}\}, x_1\rangle \langle x_2, [4,4]\rangle \langle \{\mathsf{A}, \mathsf{C}\}, [2,7]\rangle$. In particular, we can write the query string from Example 10 as $\langle x, [1,2]\rangle \langle (\mathsf{FAI}|\mathsf{KIL}), 1\rangle \langle x, [9,10]\rangle \langle \mathsf{FIN}, 1\rangle$.

We define $\mathbb{V}(q)$, $\mathbb{D}(q)$, and $\mathbb{I}(q)$ to be the set of all values, variables, disjunctions, and intervals that occur in the query string s. Formally, we let

$$\mathbb{V}(q) \coloneqq \{ x \in \mathbb{V} : \text{there ex. } (i,j) \in [\ell]\times[k] \text{ s.t. } s[i][j] = x \},$$
$$\mathbb{D}(q) \coloneqq \{ \chi \in \mathbb{D} : \text{there ex. } (i,j) \in [\ell]\times[k] \text{ s.t. } s[i][j] = \chi \},$$
$$\mathbb{I}(q) \coloneqq \{ [p,q] \in \mathbb{I} : \text{there ex. } (i,j) \in [\ell]\times[k] \text{ s.t. } s[i][j] = [p,q] \}.$$

It is reasonable to assume that each disjunctive clause in the query string is a *proper* subset of its domain, since otherwise we could also use a wildcard instead of a disjunction. Furthermore, note that setting all gap-size constraints of a dswg$^{\leqslant}$-query q to $(0,\infty)$ corresponds to a query without gap-size constraints. We call q an mdswg$^{\leqslant}$-query if q is a k-dswg$^{\leqslant}$-query for an arbitrary dimension $k \in \mathbb{N}_{\geqslant 1}$.

3.1 Query Matching

Intuitively, a k-trace t matches a k-dswg$^{\leqslant}$-query $q = (s, w, c)$ if

- each occurrence of a closed interval $[p,q]$ in s can be mapped to a value $v \in [p,q]$,
- each occurrence of a disjunctive clause χ in s can be mapped to a single type $\gamma \in \chi$, and
- the variables in s can be replaced by values of the corresponding domain,

such that the resulting string s' occurs as a subsequence in t, while satisfying the global window size w and the local gap-size constraints c. I.e., t contains a factor t' of length at most w such that s' occurs as a subsequence in t', and for each $i < \ell \coloneqq |s|$, 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 $s'[1]\, g_1\, s'[2]\, g_2 \cdots g_{\ell-1}\, s'[\ell]$ for some $g_1, \ldots, g_{\ell-1} \in \Sigma^*$ and $c_i^- \leqslant |g_i| \leqslant c_i^+$ for all $i \in [\ell{-}1]$. We formalise this via *substitutions* as follows.

Definition 11. A *substitution of size ℓ and k* in the context of k-dswg$^{\leqslant}$-queries is a mapping $\mu_{\ell,k}\colon \big(([\ell]\times[k]) \times (\mathbb{V}\cup\mathbb{I}\cup\mathbb{D})\big) \to \big(\mathbb{V}\cup\mathbb{I}\cup\mathbb{D}\big)$ with

$$\mu_{\ell,k}((i,j),z) \coloneqq \begin{cases} f_\mu(z) & \text{if } z \in \mathbb{V}, \\ [p,q] & \text{if } z \in \mathbb{I} \text{ and } [p,q] \subseteq z, \\ z' & \text{else, with } \varnothing \subset z' \subseteq z, \end{cases}$$

where $f_\mu\colon \mathbb{V} \to \big(\mathbb{V}\cup\mathbb{I}\cup\mathbb{D}\big)$ is a function. ⌟

Note that f_μ ensures that every occurrence of a variable $z \in \mathbb{V}$ is mapped to the same value from $\big(\mathbb{V}\cup\mathbb{I}\cup\mathbb{D}\big)$. In contrast, any two occurrences of a set $\chi \in \mathbb{D}$ or interval $[p,q] \in \mathbb{I}$ can be mapped to different subsets of χ or subintervals of $[p,q]$, respectively.

A substitution $\mu_{\ell,k}$ is called *final*, if for all $i,j \in [\ell]\times[k]$ and all $z \in \big(\mathbb{V}\cup\mathbb{I}\cup\mathbb{D}\big)$ it holds that $\mu_{\ell,k}((i,j),z)$ is a singleton set or a singleton interval. A final substitution can be thought of as "resolving sets and intervals", it will allow us to give a succinct definition of query matching, and it will also be useful in proofs later.

For a string $s \in \big((\mathbb{V}\cup\mathbb{I}\cup\mathbb{D})^k\big)^+$ and a substitution $\mu_{\ell,k}$ we let

$$\begin{aligned} \mu_{\ell,k}(s) \;=\; & \big\langle \mu_{\ell,k}((1,1),s[1][1]),\ \ldots,\ \mu_{\ell,k}((1,k),s[1][k])\big\rangle\ \ldots \\ & \big\langle \mu_{\ell,k}((\ell,1),s[\ell][1]),\ \ldots,\ \mu_{\ell,k}((\ell,k),s[\ell][k])\big\rangle. \end{aligned}$$

Note that if we apply a substitution $\mu_{\ell,k}$ to a string, we assume that the entire length of the string (including attributes) is $\ell \cdot k$. Hence, we can omit the indices ℓ and k. Furthermore, we may omit the parameter (i,j) if the position is given by the context, or if the second parameter is a variable, since variables are always mapped to the same value, i. e., since $\mu((i,j),x) = \mu((i',j'),x)$ for all $i,i' \in [\ell]$ and $j,j' \in [k]$ holds for all $x \in \mathbb{V}$.

An embedding $\xi\colon [\ell] \to [n]$ is said to *satisfy a global window size w*, if $\xi(\ell) - \xi(1) + 1 \leqslant w$. We say ξ *satisfies a tuple $c = (c_1, c_2, \ldots, c_{\ell-1})$ of local gap-size constraints* (for ℓ and w) if $c_i^- \leqslant \xi(i{+}1) - 1 - \xi(i) \leqslant c_i^+$ for all $i < \ell$.

We are now ready to give a precise definition of when a query matches in a trace.

Definition 12. A k-dswg$^{\leqslant}$-query $q = (s,w,c)$ *matches in a k-trace t* (or simply *t matches q*), if and only if there is a *final* substitution μ and an embedding $\xi\colon [\ell] \to [|t|]$ that satisfies w and c, such that for all $i,j \in [\ell]\times[k]$ it holds that $t[\xi(i)][j] \in s[i][j]$. Note that since μ is final, this is equivalent to $s[i][j] = \big[t[\xi(i)][j], t[\xi(i)][j]\big]$ or $s[i][j] = \big\{\, t[\xi(i)][j] \,\big\}$, depending on whether A_j is ordered or not. We call (μ,ξ) *a witness for $t \models q$*. ⌟

Example 13. Consider the schema $\mathcal{A}_{\text{ex}}$ from Example 8 and the 2-dswg$^{\leqslant}$-query q from Example 10. We observe that the traces t_1, t_2, and t_3 match q. For example, for $t_1 \models q$, a witness can be illustrated as follows:

$$\begin{array}{rcccccc} s &=& \langle x, [1,2]\rangle & \langle(\mathsf{FAI}|\mathsf{KIL}), 1\rangle & \langle x, [9,10]\rangle & & \langle \mathsf{FIN}, 1\rangle \\ t_1 &=& \langle \mathsf{SCH}, 1\rangle & \langle \mathsf{FAI}, 1\rangle & \langle \mathsf{SCH}, 9\rangle & \langle \mathsf{UPD}, 9\rangle & \langle \mathsf{FIN}, 1\rangle \end{array}$$

However, if we consider the trace $t_4 \coloneqq \langle \mathsf{FAI}, 5\rangle\ \langle \mathsf{KIL}, 1\rangle\ \langle \mathsf{SUB}, 1\rangle\ \langle \mathsf{SCH}, 1\rangle\ \langle \mathsf{FAI}, 1\rangle$ it turns out that $t_4 \not\models q$, since no substitution μ can be found such that $\mu(s)$ can be embedded into t_4, let alone that a corresponding embedding would satisfy w and c. ⌟

A k-dswg$^{\leqslant}$-query $q = (s, w, c)$ is also called a (k, ℓ, w, c)-*dswg*$^{\leqslant}$-*query* for $\ell = |s|$. If the maximal size of a disjunction occurring in a (k, ℓ, w, c)-dswg$^{\leqslant}$-query q is bounded by a number $d \geqslant 1$, we call q a (k, ℓ, w, c, d)-*dswg*$^{\leqslant}$-*query*. We will refer to (k, ℓ, w, c, d) as *query parameters*. For arbitrary dimension $k \in \mathbb{N}_{\geqslant 1}$ we simply call a k-dswg$^{\leqslant}$-query q a *mdswg*$^{\leqslant}$-*query*.

3.2 Query Containment

This section is devoted to a characterisation of containment and equivalence of (k, ℓ, w, c, d)-dswg$^{\leqslant}$-queries via suitable notions of homomorphisms and isomorphisms. For this section, let us fix a query length ℓ, a window size $w \in \mathbb{N} \cup \{\infty\}$, a tuple c of local gap-size constraints for ℓ and w, and a maximum d for the size of disjunctions.

The *model set* of a k-dswg$^{\leqslant}$-query q is defined as

$$\mathsf{Mod}(q) \coloneqq \left\{ t \in \big(dom(A_1) \times \cdots \times dom(A_k)\big)^+ \,:\, t \models q \right\}.$$

We say that q *is contained in* q' if $\mathsf{Mod}(q) \subseteq \mathsf{Mod}(q')$, and we say that q *and* q' *are equivalent* if $\mathsf{Mod}(q) = \mathsf{Mod}(q')$.

Definition 14. Let q and q' be (k, ℓ, w, c, d)-dswg$^{\leqslant}$-queries. A homomorphism from q' to q is a substitution h such that $h(s') = s$ and the following property holds:

- For every $z \in \mathbb{V}$ that occurs at least twice in the query string s' of q' that is *not* mapped to a variable, we have that $h(z)$ is a singleton set or a singleton interval.

We write $q' \xrightarrow{\text{hom}} q$ to express that there exists a homomorphism from q' to q. ⌟

Example 15. As a justification of this property, consider the simple 2-dswg$^{\leqslant}$-queries $q_1 \coloneqq (s_1, w, c)$, $q_2 \coloneqq (s_2, w, c)$ over the schema $\mathcal{A}_{\text{ex}}$ from Example 8 with suitable window size w and local gap-size constraints c and the query strings $s_1 = \langle x, 1\rangle\,\langle x, 1\rangle$ and $s_2 = \langle(\mathsf{SCH} \mid \mathsf{KIL}), 1\rangle\,\langle(\mathsf{SCH} \mid \mathsf{KIL}), 1\rangle$. If we did not impose the above requirement, a substitution mapping x to $(\mathsf{SCH} \mid \mathsf{KIL})$ would be a homomorphism. However, this clearly does not preserve the "structure" of q_1, since the trace $\langle \mathsf{SCH}, 1\rangle\,\langle \mathsf{KIL}, 1\rangle$ matches q_2, but not q_1. ⌟

The following theorem characterises containment via homomorphisms.

Theorem 16. *For any (k, ℓ, w, c, d)-dswg$^{\leqslant}$-queries q and q' it holds that*

$$\mathsf{Mod}(q) \subseteq \mathsf{Mod}(q') \iff q' \xrightarrow{\text{hom}} q.$$

Proof. For the backward direction "$\Leftarrow$" we have to show the following claim. Since the proof is lengthy but rather straightforward, we defer it Appendix A.

Claim. *Let $q' \xrightarrow{\text{hom}} q$ and let t be a trace with $t \models q$. Then $t \models q'$.*

We show the forward direction "$\Rightarrow$" via the following claim.

Claim. *Let $\mathsf{Mod}(q) \subseteq \mathsf{Mod}(q')$. Then any substitution $h\colon \big(([\ell]\times[k]) \times (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})\big) \to (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ with $h((i,j), s'[i][j]) = s[i][j]$ for all $(i,j) \in [\ell]\times[k]$ is a homomorphism from q' to q.*

Proof of Claim. For every final substitution $\mu\colon \big(([\ell]\times[k]) \times (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})\big) \to (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ let $\gamma_{\mu,i,j}$ be the symbol such that $\mu((i,j), s[i][j]) = \{\gamma_{\mu,i,j}\}$ or $[\gamma_{\mu,i,j}, \gamma_{\mu,i,j}]$, respectively. Further, for each $i \in [\ell]$ let $e_{\mu,i} \coloneqq \langle \gamma_{\mu,i,1}, \gamma_{\mu,i,2}, \ldots, \gamma_{\mu,i,k} \rangle$ and let g_i be a sequence of c_i^- copies of a fixed symbol $\gamma \in \Gamma$ for every $i \in [\ell - 1]$. Finally, let $t_\mu \coloneqq e_{\mu,1} g_1 e_{\mu,2} g_2 \cdots g_{\ell-1} e_{\mu,\ell}$.

Clearly, for every such substitution μ it holds that $t_\mu \models q$ as witnessed by (μ, ξ) with $\xi(i) = i + \sum_{j<i} c_j^-$ for all $i \in [\ell]$. Since $\mathsf{Mod}(q) \subseteq \mathsf{Mod}(q')$ it holds that $t_\mu \models q'$ and by the choice of the gaps g_i, every witness (μ', ξ') must satisfy $\xi'(i) = i + \sum_{j<i} c_i^-$, i. e., $\mu'(s'[i]) = \mu(s[i])$ for all $i \in [\ell]$.

We have to show that h defined by $h((i,j), s'[i][j]) \coloneqq s[i][j]$ is a homomorphism from q' to q. Let $i \in [\ell]$ and $j \in [k]$. We distinguish the following cases.

Case 1: $s'[i][j]$ is of the form $[p, q] \in \mathbb{I}_j$. Assuming that $s[i][j]$ is not a subinterval of $[p, q]$, there must be a final substitution μ with $\mu((i,j), s[i][j]) = [\gamma, \gamma]$ for some $\gamma \notin [p, q]$ such that $t_\mu \models q$. However, since $\gamma \notin [p, q]$, $t_\mu \not\models q'$. This is contradictory, hence h maps $s'[i][j]$ to a subinterval of $[p, q]$.

Case 2: $s'[i][j]$ is of the form $\chi \in \mathbb{D}_j$. Assuming that $s[i][j]$ is not a subset of χ, there must be a final substitution μ with $\mu((i,j), s[i][j]) = \{\gamma\}$ for some $\gamma \notin \chi$ such that $t_\mu \models q$. However, since $\gamma \notin \chi$, $t_\mu \not\models q'$. This is contradictory, hence h maps $s'[i][j]$ to a subset of χ.

Case 3: $s'[i][j]$ is of the form $x \in \mathbb{V}$. Since $h((i,j), s'[i][j]) = s[i][j], x$ is either mapped to a variable, an interval or a subset of $dom(A_j)$. If there is no other pair $(\hat{\imath}, \hat{\jmath}) \neq (i,j)$ such that $s'[\hat{\imath}][\hat{\jmath}] = x$, then we are done with this case. Otherwise, if $h((i,j), s'[i][j]) =: y \neq z \coloneqq h((\hat{\imath}, \hat{\jmath}), s'[\hat{\imath}][\hat{\jmath}])$, then a final substitution μ with $\mu((i,j), y) \neq \mu((\hat{\imath}, \hat{\jmath}), z)$ would

yield a trace t_μ where $t_\mu \models q$ but $t_\mu \not\models q'$, which would contradict $\mathsf{Mod}_\Gamma(q) \subseteq \mathsf{Mod}_\Gamma(q')$. Thus, $h(s'[i][j]) = h(s'[\hat{\imath}][\hat{\jmath}])$ must hold.
If they map to a variable, we are done. If they map to a set χ, then we must show that χ is singleton. Assuming that this were not the case, consider a final substitution μ with $\mu((i,j),\chi) = \{\gamma\}$ and $\mu((\hat{\imath},\hat{\jmath}),\chi) = \{\gamma'\}$ for $\gamma \neq \gamma' \in \chi$. Again, this would lead us to $t_\mu \models q$, but $t_\mu \not\models q'$, which is a contradiction. Hence, χ is singleton. If they map to an interval, the same argument can be used to show that it must be singleton as well. □

3.3 Query Equivalence and Isomorphisms

To conclude Section 3, let us discuss the characterisation of equivalence for two queries via isomorphisms. We defer the proofs to Appendix A.

Definition 17. Two (k,ℓ,w,c,d)-dswg$^{\leqslant}$-queries q and q' are called *isomorphic* (denoted by $q \cong q'$), if there is a bijection $\pi\colon (\mathbb{V}(q) \cup \mathbb{I} \cup \mathbb{D}) \to (\mathbb{V}(q') \cup \mathbb{I} \cup \mathbb{D})$, such that $\pi_{|(\mathbb{I}\cup\mathbb{D})} = \mathrm{id}$ and $\pi(s[i][j]) = s'[i][j]$ for all $(i,j) \in [\ell]\times[k]$. ⌟

Definition 18. Let q and q' be two (k,ℓ,w,c,d)-dswg$^{\leqslant}$-queries and $I \subseteq [\ell]\times[k]$. We say that q is partially isomorphic to q' w.r.t I (denoted by $q \sim_I q'$) if, and only if,

1. for all $(i,j),(i',j') \in I$ we have $s[i][j] = s[i'][j'] \iff s'[i][j] = s'[i'][j']$, and
2. for all $(i,j) \in I$ we have:
   $s[i][j] \in (\mathbb{I}\cup\mathbb{D})$ iff $\big(s'[i][j] \in (\mathbb{I}\cup\mathbb{D})$ and $s[i][j] \in (\mathbb{I}\cup\mathbb{D}) \Rightarrow s[i][j] = s'[i][j]\big)$. ⌟

Lemma 19. *For any (k,ℓ,w,c,d)-dswg$^{\leqslant}$-queries q and q' it holds that*
$q \cong q' \iff q \xrightarrow{\text{hom}} q'$ *and* $q' \xrightarrow{\text{hom}} q \iff \mathsf{Mod}(q) = \mathsf{Mod}(q')$.

Lemma 20. *For all (k,ℓ,w,c,d)-dswg$^{\leqslant}$-queries q, q' we have $q \sim_{[\ell]\times[k]} q' \iff q \cong q'$.*

4 Query Discovery

Next, we turn to the question of how meaningful mdswg$^{\leqslant}$-queries can be discovered from a given set of traces. For mswg-queries and dswg-queries, this question was answered algorithmically in [13] and [8], respectively.

A *k-dimensional sample over $\mathcal{A}$* is a finite, non-empty set $\mathcal{S}$ of k-traces over $\mathcal{A}$. We denote the set of all values occurring in $\mathcal{S}$ by $adom(\mathcal{S})$, i. e., $adom(\mathcal{S}) = \bigcup_{t\in\mathcal{S}} adom(t) \subseteq \Gamma$. Let $j \in [k]$ be an attribute index. Then we denote the set of all values that occur in some event of a trace $t \in \mathcal{S}$ at position j by $adom_j(\mathcal{S})$. For the rest of this section, we restrict the domains of $\mathcal{A}$ to $\mathcal{S}$, i. e., we assume that $dom(A_j) = adom_j(\mathcal{S})$ for all $j \in [k]$. This will be convenient for the proofs.

The *support* $\mathsf{supp}(q,\mathcal{S})$ of a query q in $\mathcal{S}$ is defined as the ratio of traces in the sample that match q to the number of traces in $\mathcal{S}$, i. e., $\mathsf{supp}(q,\mathcal{S}) \coloneqq \frac{|\{\, t\in\mathcal{S} : t\models q \,\}|}{|\mathcal{S}|}$. 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 $\mathsf{supp}(q, \mathcal{S}) \geqslant \mathsf{sp}$.

Before turning to the definition of meaningful mdswg$^{\leqslant}$-queries w.r.t. to a given k-dimensional sample $\mathcal{S}$ and support threshold sp, let us explain this notion on an intuitive level for $\mathsf{sp} = 1.0$. A meaningful k-dswg$^{\leqslant}$-query q for this scenario should be matched by all traces in the sample, i. e. $\mathcal{S} \subseteq \mathsf{Mod}(q)$. Furthermore, it should be one of the best descriptors for $\mathcal{S}$ in the sense that there is no other query q' that can be "wrapped around" $\mathcal{S}$ more tightly, i. e., in the sense that $\mathcal{S} \subseteq \mathsf{Mod}(q') \subsetneq \mathsf{Mod}(q)$. Note that meaningful queries shall be defined via query containment. Since the latter requires to fix the query parameters (k, ℓ, w, c, d), they also have to be fixed in the following definition.

Definition 21. Let $\mathcal{S}$ be a k-dimensional sample over $\mathcal{A}$. Let sp be a support threshold and $d \in \mathbb{N}_{\geqslant 1}$. A (k, ℓ, w, c, d)-query q (over $\mathcal{A}$) is called *descriptive for* $\mathcal{S}$ *w.r.t.* $\mathcal{A}$, sp, *and* (k, ℓ, w, c, d) if $\mathsf{supp}(q, \mathcal{S}) \geqslant \mathsf{sp}$, and there is no other (k, ℓ, w, c, d)-query q' (over $\mathcal{A}$) with $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ and $\mathsf{Mod}(q') \subsetneq \mathsf{Mod}(q)$. ⌟

Note that d should be bounded by $\max_{j \in [k]}(|adom_j(\mathcal{S})|) - 1$ since otherwise we could use a wildcard instead. When considering intervals and disjunctions together, note that a disjunction is as or even more restrictive than an interval, since every interval can also be seen as the disjunction of the values it contains, while this does not hold the other way round. As an example, consider a schema $\mathcal{A} = (A_1)$ with $dom(A_1) = \mathbb{N}_{\geqslant 1}$. Then a query $q = (s, w, c)$ over $\mathcal{A}$ with $s = (i|j)$ for $i, j \in \mathbb{N}_{\geqslant 1}$ is strictly more restrictive than or as restrictive as a query $q' = (s, w, c)$ over $\mathcal{A}$ with $s = [i, j]$, if $i + 1 < j$ or $i \leqslant j \leqslant i + 1$, respectively. Therefore, defining descriptive queries w.r.t. to a fixed event schema (among others) and thereby disallowing disjunctions and intervals on the same attribute is vital when turning towards discovery of descriptive queries that allow for intervals and disjunctions at the same time.

Example 22. Consider the schema $\mathcal{A}_{\text{ex}}$ from Example 8, the sample from Example 22

$$\begin{aligned}\mathcal{S} \coloneqq \big\{ &\langle \mathsf{SCH}, 1\rangle \langle \mathsf{FAI}, 1\rangle \langle \mathsf{SCH}, 9\rangle \langle \mathsf{UPD}, 9\rangle \langle \mathsf{FIN}, 1\rangle \\ &\langle \mathsf{UPD}, 2\rangle \langle \mathsf{FAI}, 1\rangle \langle \mathsf{UPD}, 10\rangle \langle \mathsf{FIN}, 1\rangle \\ &\langle \mathsf{SUB}, 1\rangle \langle \mathsf{KIL}, 1\rangle \langle \mathsf{SUB}, 9\rangle \langle \mathsf{SCH}, 8\rangle \langle \mathsf{FIN}, 1\rangle \big\},\end{aligned}$$

and let $x, x_1, x_2 \in \mathbb{V}$. Furthermore, we consider the query parameters $k = 2$, $\ell = 4$, $w = 5$, $c = ((0, 0), (0, 0), (0, 1))$, and $d = 2$, and the support threshold $\mathsf{sp} = 1.0$.

A query of form $(\langle x, [1, 2]\rangle \langle(\mathsf{FAI}|\mathsf{KIL}), 1\rangle \langle x, [9, 10]\rangle \langle \mathsf{FIN}, 1\rangle, w, c)$ (as in Example 10) is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}_{\text{ex}}$, sp, and (k, ℓ, w, c, d). To see that this is indeed the case, first note that $\mathsf{supp}(q, \mathcal{S}) \geqslant \mathsf{sp}$, since all traces in $\mathcal{S}$ match q, as discussed in Example 13. Furthermore, the only further refinements of q would be the replacement of

- x by a disjunction χ over $dom(A_1)$ of size 2,
- $[1, 2]$ by an interval $I \subset [1, 2]$,

- (FAI|KIL) by either FAI or KIL, or
- $[9, 10]$ by an interval $I \subset [9, 10]$,

but the resulting query q' would not satisfy the support threshold any more. Hence, q is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}_{\text{ex}}$, sp, and (k, ℓ, w, c, d).

The most general query for the given query parameters is

$$q_{mg} \quad = \quad (\langle x_{1,1}, x_{1,2}\rangle\langle x_{2,1}, x_{2,2}\rangle\langle x_{3,1}, x_{3,2}\rangle, w, c).$$

Note that neither q_{mg} nor the query

$$q'' \quad \coloneqq \quad (\langle x_1, [1, 2]\rangle \langle (\text{FAI}|\text{KIL}), 1\rangle \langle x_2, [9, 10]\rangle \langle \text{FIN}, 1\rangle, w, c)$$

are descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}_{\text{ex}}$, sp, and (k, ℓ, w, c, d), as $\mathsf{Mod}(q) \subsetneq \mathsf{Mod}(q_{mg})$ and $\mathsf{Mod}(q) \subsetneq \mathsf{Mod}(q'')$. ⌟

Remark 23. Recall that we assume that $|dom(A_j)| \geqslant 2$ for all $j \in [k]$. Hence, by Theorem 16 and Lemma 19 we know that an mdswg$^{\leqslant}$-query q is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}$, sp, and (k, ℓ, w, c, d) if, and only if, q is a (k, ℓ, w, c, d)-query over $\mathcal{A}$, $\mathsf{supp}(q, \mathcal{S}) \geqslant \mathsf{sp}$, and there is no (k, ℓ, w, c, d)-query q' over $\mathcal{A}$ with $q \not\equiv q'$ such that $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ and $q \xrightarrow{\text{hom}} q'$.

Let $A_j \in \mathcal{A}$ with $j \in U$ be an attribute. Recall that we assumed $dom(A_j) = adom_j(\mathcal{S})$. We write $\mathbb{D}_j$ (or $\mathbb{D}_j(adom(\mathcal{S}))$) to denote all finite non-empty subsets over $dom(A_j)$ (or $adom_j(\mathcal{S})$, respectively). A disjunction $\chi \in \mathbb{D}_j(adom(\mathcal{S}))$ (or an interval $I \in \mathbb{I}$) *satisfies* sp *w.r.t. to* $\mathcal{S}$ *and* A_j, if the fraction of traces in $\mathcal{S}$ containing some $\gamma \in \chi$ (or $\gamma \in I$, respectively) at attribute A_j is greater than or equal to sp. The set of all disjunctions that satisfy sp w.r.t. to $\mathcal{S}$ and A_j is

$$\Delta_{A_j}(\mathcal{S}, \mathsf{sp}) \coloneqq \{\, \chi \in \mathbb{D}_j(adom(\mathcal{S})) \;:\; \frac{\{\, t \in \mathcal{S} \;:\; \text{ex. } i \in [|t|] \text{ s.t. } t[i][j] \in \chi \,\}}{|\mathcal{S}|} \geqslant \mathsf{sp} \,\}.$$

We omit $\mathcal{S}$ and sp, if they are clear from the context. Furthermore, we write Δ to denote the set of all disjunctions over all attribute indices that satisfy sp, i. e. $\Delta = \bigcup_{j \in U, i \in \mathbb{N}_{\geqslant 1}} \Delta^i_{A_j}$. For $d \in \mathbb{N}_{\geqslant 1}$ we write $\Delta^d_{A_j}$ to denote the subset of Δ which contains only disjunctions of size precisely d over A_j. For a descriptive (k, ℓ, w, c, d)-query q it holds that $\mathbb{D}(q) \subseteq \Delta^1 \dot\cup \ldots \dot\cup \Delta^d$. This coincides with the notion of non-disjunctive mswg-queries presented in [12, 13] since $\Delta = \Delta^1 = \{\, \{\gamma\} \in \mathbb{D} \;:\; \frac{|\{\, t \in \mathcal{S} \,:\, \gamma \in adom(t) \,\}|}{|\mathcal{S}|} \geqslant \mathsf{sp} \,\}$, can be used to discover non-disjunctive mswg-queries.

Given a (k, ℓ, w, c, d)-query $q = (s, w, c)$, a variable $x \in \mathbb{V}(q)$ and a symbol $z \in (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ we write $s\langle x \mapsto z\rangle$ to denote the query string which is obtained from s by replacing every occurrence of x in s by z. Next, we present the algorithmical idea for query discovery.

Compute Descriptive Query Problem (CompDescQuery): On input of a k-dimensional sample $\mathcal{S}$ over $\mathcal{A}$, a support threshold sp, a string length $\ell \in \mathbb{N}_{\geqslant 1}$, a global window size $w \geqslant \ell$, a tuple c of gap-size constraints, and $d \in \mathbb{N}_{\geqslant 1}$, the task is to compute a (k, ℓ, w, c, d)-query q that is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}$, sp, and (k, ℓ, w, c, d).

Pseudocode of an algorithm solving CompDescQuery is provided in Algorithm 1. Given a k-dimensional sample $\mathcal{S}$ over $\mathcal{A}$, a support threshold sp and query parameters (k, ℓ, w, c, d) as input, the algorithm first builds the *most general query* $q = q_{mg}$ for (k, ℓ, w, c, d), in the sense that $\mathsf{Mod}(q') \subseteq \mathsf{Mod}(q_{mg})$ for each (k, ℓ, w, c, d)-query q'. Its query string consists of $k \cdot \ell$ distinct variables, i. e. $s_{mg} = \langle x_{1,1}, \ldots, x_{1,k} \rangle \cdots \langle x_{\ell,1}, \ldots, x_{\ell,k} \rangle$.

If $\mathsf{supp}(q, \mathcal{S}) < \mathsf{sp}$ the algorithm stops and returns $\perp$ (Line 2 of Algorithm 1), because no other query q' with $\mathsf{Mod}(q') \subseteq \mathsf{Mod}(q_{mg})$ can describe $\mathcal{S}$ with sufficient support.

Otherwise, the algorithm searches for an admissible *replacement operation* for each variable $x_{m,n} \in V^{\mathrm{u}} := \mathbb{V}(s) = \{x_{1,1}, \ldots, x_{\ell,k}\}$ during the main loop of Algorithm 1 (Line 6). A replacement operation replaces $x_{m,n}$ by

Algorithm 1: CompDescQuery($\mathcal{S}$, sp, (k, ℓ, w, c, d))

Input : sample $\mathcal{S}$; support threshold sp with $0 < \mathsf{sp} \leqslant 1$; (k, ℓ, w, c, d)
Returns : descriptive query q for $\mathcal{S}$ w.r.t. $\mathcal{A}$, sp, and (k, ℓ, w, c, d) or error message $\perp$

1 $s := s_{mg}$; $q := (s_{mg}, w, c)$ // query string and query
2 **if** $\mathsf{supp}(q, \mathcal{S}) < \mathsf{sp}$ **then stop and return** $\perp$
3 $\Delta := \bigcup_{n \in U} \Delta^1_{A_n} \dot\cup \ldots \dot\cup \bigcup_{n \in U} \Delta^d_{A_n}$ // disjunctions to be considered
4 $\nabla := \{n \to sort(adom_n(\mathcal{S})) \mid n \in O\}$ // indices to values to be considered
5 $V^{\mathrm{u}} := \mathbb{V}(q)$; $V^{\mathrm{a}} := \emptyset$ // unvisited and available variables
6 **while** $V^{\mathrm{u}} \neq \emptyset$ **do**
7 **select** an arbitrary $x_{m,n} \in V^{\mathrm{u}}$ and let $V^{\mathrm{u}} := V^{\mathrm{u}} \setminus \{x_{m,n}\}$
8 replace := False
9 **if** $n \in U$ **then** // $dom(A_n)$ is unordered
10 $\Delta^1_n := \Delta^1_n \cup V^{\mathrm{a}}$
11 **for** $i = 1$ *to* $\min(d, |adom_n(\mathcal{S})| - 1)$ **do**
12 $\Omega := \Delta^i_n$
13 q, replace := CheckRepl($\mathcal{S}$, sp, q, $x_{m,n}$, Ω, replace) // try DisjOrVarRepl
14 **if** replace *is* True **then break**
15 **else** // $dom(A_n)$ is ordered
16 $\Omega := V^{\mathrm{a}}$
17 q, replace := CheckRepl($\mathcal{S}$, sp, q, $x_{m,n}$, Ω, replace) // try VarRepl
18 **if** replace *is* False **then**
19 $\varrho := \nabla(n)$
20 q, replace := CheckInterRepl($\mathcal{S}$, sp, q, $x_{m,n}$, ϱ, replace) // try InterRepl
21 **if** replace *is* False **then**
22 $V^{\mathrm{a}} := V^{\mathrm{a}} \cup \{x_{m,n}\}$ // NoChangeOp
23 **stop and return** $q := (s, w, c)$

- an "available" variable $x \in V^{\mathrm{a}}$, called *VarOp*, (Line 17 or during the first iteration of the for-loop in Line 13 of Algorithm 1),
- a disjunction $y \in \Delta_n^i$ for $i \geqslant 1$, called *DisjOp*, (Line 13 calls Function 2), or
- an interval $I \subset dom(A_n)$, called *InterOp*, (Line 20 calls Function 3).

A variable is called *available* if it has been considered before during the run of the algorithm and has not been replaced. Note that both, intervals and disjunctions, may be singletons. Particularly, in the case of an unordered domain, singletons are checked first for replacement operations.

The replacement operation is stored in q and called *admissible* if the resulting query satisfies the support threshold (Line 4 of Function 2, and Lines 6, 8, and 10 of Function 3). Note that in case of a replacement by an interval I this interval is minimal in the sense that no interval

Function 2: CheckRepl($\mathcal{S}$, sp, q, $x_{m,n}$, Ω, replace)

Input : sample $\mathcal{S}$; support threshold sp; query q; variable $x_{m,n}$; available symbols Ω; flag replace
Returns : query q; flag replace

1 **while** $\Omega \neq \emptyset$ **do**
2 **select** an arbitrary $y \in \Omega$ and let $\Omega := \Omega \setminus \{ y \}$
3 $q' := (s\langle x_{m,n} \mapsto y \rangle, w, c)$
4 **if** supp$(q', \mathcal{S}) \geqslant$ sp **then**
5 $s := s\langle x_{m,n} \mapsto y \rangle$; replace := True // ReplaceOp
6 **return** q; replace
7 **return** q; replace

Function 3: CheckInterRepl($\mathcal{S}$, sp, q, $x_{m,n}$, ϱ, replace)

Input : sample $\mathcal{S}$; support threshold sp; query q; variable $x_{m,n}$; values $\varrho \subseteq dom(A_n)$ occurring in $\mathcal{S}$; flag replace
Returns : query q; flag replace

1 $pos = [1, \ldots, |\varrho|]$ // list of positions over values in ϱ
2 low := 1; up := $|\varrho|$ // pointer to current interval bounds
3 lowb := up + 1; upb := low − 1 // restrictions for interval bound pointer
4 **while** *lowb* − *low* > 1 *or* *up* − *upb* > 1 **do**
5 nlow := low + $\lceil \frac{|\text{low}-\text{lowb}|}{2} \rceil$; nup := up − $\lfloor \frac{|\text{up}-\text{upb}|}{2} \rfloor$
6 **if** supp$((s\langle x_{m,n} \mapsto [\varrho[\textit{nlow}], \varrho[\textit{nup}]] \rangle, w, c), \mathcal{S}) \geqslant$ sp **then**
7 low := nlow; up := nup
8 **else if** supp$((s\langle x_{m,n} \mapsto [\varrho[\textit{nlow}], \varrho[\textit{up}]] \rangle, w, c), \mathcal{S}) \geqslant$ sp **then**
9 low := nlow; upb := nup
10 **else if** supp$((s\langle x_{m,n} \mapsto [\varrho[\textit{low}], \varrho[\textit{nup}]] \rangle, w, c), \mathcal{S}) \geqslant$ sp **then**
11 lowb := nlow; up := nup
12 **else** lowb := nlow; upb := nup
13 **if** *low* == min(ϱ) *and* *up* == max(ϱ) **then return** q; False
14 **else return** $(s\langle x_{m,n} \mapsto [\varrho[\text{low}], \varrho[\text{up}]] \rangle, w, c)$; True // ReplaceOp

$I' \subset I$ at the current position in the query string would lead to a query q' that also satisfies the support threshold sp (see Function 3).

In case that no admissible replacement operation was found, the query string remains unchanged, and $x_{m,n}$ gets available (Line 22 of Algorithm 1). We call this operation *NoChangeOp*. After each variable in $\mathbb{V}(s_{mg})$ has been considered, the algorithm terminates and returns the current query as output (Line 23 of Algorithm 1).

Next we depict an exemplaric run of Algorithm 1. We refer to [7] for a brief discussion, why Δ is passed through incrementally in Line 11. Intuitively speaking, replacing a variable x in a query string by some disjunction χ may lead to a non-descriptive query if there exists a disjunction $\chi' \subset \chi$ that would also be an admissible replacement of x. This can be averted by passing through Δ incrementally.

Example 24. We take up Example 13 and consider $\mathcal{A}_{\text{ex}} = (\mathsf{task}, \mathsf{priority})$ with $dom(\mathsf{task}) = \{\mathsf{SUB}, \mathsf{SCH}, \mathsf{UPD}, \mathsf{KIL}, \mathsf{FAI}, \mathsf{FIN}\}$ and $dom(\mathsf{priority}) = \{1, \ldots, 10\}$, whereby $dom(\mathsf{priority})$ is ordered and 1 and 10 are the minimal and maximal elements in $dom(\mathsf{priority})$ w.r.t. the order. Furthermore, we consider the sample

$$\begin{aligned}\mathcal{S} \coloneqq \big\{ &\langle \mathsf{SCH}, 1\rangle\ \langle \mathsf{FAI}, 1\rangle\ \langle \mathsf{SCH}, 9\rangle\ \langle \mathsf{UPD}, 9\rangle\ \langle \mathsf{FIN}, 1\rangle \\ &\langle \mathsf{UPD}, 2\rangle\ \langle \mathsf{FAI}, 1\rangle\ \langle \mathsf{UPD}, 10\rangle\ \langle \mathsf{FIN}, 1\rangle \\ &\langle \mathsf{SUB}, 1\rangle\ \langle \mathsf{KIL}, 1\rangle\ \langle \mathsf{SUB}, 9\rangle\ \langle \mathsf{SCH}, 8\rangle\ \langle \mathsf{FIN}, 1\rangle \big\},\end{aligned}$$

the query parameters $k = 2$, $\ell = 4$, $w = 5$, $c = ((0,0),(0,0),(0,1))$, and $d = 2$, and the support threshold $\mathsf{sp} = 1.0$. On input of $(\mathcal{S}, \mathsf{sp}, (k, \ell, w, c, d))$ the algorithm first generates

$$q = (\langle x_{1,1}, x_{1,2}\rangle\langle x_{2,1}, x_{2,2}\rangle\langle x_{3,1}, x_{3,2}\rangle\langle x_{4,1}, x_{4,2}\rangle, w, c).$$

Since q satisfies the support threshold the algorithm proceeds by computing $\Delta = \{\{\mathsf{FIN}\}\} \dot\cup \{\{\mathsf{SUB}, \mathsf{UPD}\}, \{\mathsf{SUB}, \mathsf{FAI}\}, \{\mathsf{SCH}, \mathsf{UPD}\}, \{\mathsf{SCH}, \mathsf{FAI}\}, \{\mathsf{UPD}, \mathsf{KIL}\}, \{\mathsf{KIL}, \mathsf{FAI}\}\}$.

Next, the algorithm computes a mapping ∇ which associates each attribute index $n \in [k]$ (for $n \in O$) with a sorted array of all values that occur within the given sample $\mathcal{S}$ in some event at attribute A_n. For this example, ∇ maps $\mathsf{priority}$ to $[1, 2, 8, 9, 10]$.

Assume the algorithm selects $x_{2,1}$ during the first iteration of the main loop. Since $A_1 = \mathsf{task}$ and $1 \in U$, Algorithm 1 searches for a replacement by a disjunction (Lines 10–13). It turns out that $\Delta^1_{\mathsf{task}} = \{\{\mathsf{FIN}\}\}$ does not contain a disjunction for an admissible replacement of $x_{2,1}$. Hence, the algorithm continues by checking disjunctions of size $i = 2$. The only admissible replacement is $s\langle x_{2,1} \mapsto (\mathsf{FAI}|\mathsf{KIL})\rangle$ for $\{\mathsf{FAI}, \mathsf{KIL}\} \in \Omega = \Delta^2_{\mathsf{task}}$, i. e. Function 2 returns $q = (s\langle x_{2,1} \mapsto (\mathsf{FAI}|\mathsf{KIL})\rangle, w, c)$ and True while V^{a} remains empty.

Let us assume that during the second and third transition through the main loop the algorithm selects the unvisited variables $x_{2,2}$ and $x_{4,2}$. Since $A_2 = \mathsf{priority}$, $2 \in O$, and $V^{\text{a}} = \emptyset$, the algorithm calls Function 3 with $\varrho = \nabla(\mathsf{priority}) = [1, 2, 8, 9, 10]$ in Line 20. For $x_{2,2}$ the function starts with the parameters $\mathtt{low} = 1$, $\mathsf{up} = 5$, $\mathtt{lowb} = 6$, and $\mathsf{upb} = 0$. During the first

iteration of the while-loop (Line 4), the new computed pointer to the lower and upper bounds are $\texttt{nlow} = 4$ and $\texttt{nup} = 3$. The checks in Lines 6 and 8, i. e. with $s\langle x_{2,2} \mapsto [\varrho[4], \varrho[3]]\rangle$ and $s\langle x_{2,2} \mapsto [\varrho[4], \varrho[5]]\rangle = s\langle x_{2,2} \mapsto [9, 10]\rangle$ are not successful, while the query with $s\langle x_{2,2} \mapsto [\varrho[1], \varrho[3]]\rangle = s\langle x_{2,2} \mapsto [1, 8]\rangle$ satisfies the support threshold (Line 10). Hence, $\texttt{lowb}$ is set to $\texttt{nlow} = 4$ and $\texttt{up}$ is set to $\texttt{nup} = 3$. During the next iteration of the while-loop, the checks in Lines 6 and 8 for $\texttt{nlow} = 3$ and $\texttt{nup} = 2$ fail again, while $s\langle x_{2,2} \mapsto [\varrho[1], \varrho[2]]\rangle = s\langle x_{2,2} \mapsto [1, 2]\rangle$ satisfies the support threshold. Therefore, $\texttt{lowb}$ is set to $\texttt{nlow} = 3$ and $\texttt{up}$ is set to $\texttt{nup} = 2$. The function computes $\texttt{nlow} = 2$ and $\texttt{nup} = 1$ during the third iteration of the while-loop, and again, only the check in Line 10 is successful and $\texttt{lowb}$ is set to $\texttt{nlow} = 2$ while $\texttt{up}$ is set to $\texttt{nup} = 1$. It holds that $\texttt{lowb} - \texttt{low} = 1$ and $\texttt{up} - \texttt{upb} = 1$, as well as $\texttt{low} \neq \min(\varrho)$ and $\texttt{up} \neq \max(\varrho)$. Hence, the function returns the new query $(s\langle x_{2,2} \mapsto [1, 1]\rangle, w, c)$ and True.

It can be easily verified that Function 3 replaces $x_{4,2}$ by $[1, 1]$ as well. Therefore, s is replaced by $(x_{1,1}, x_{1,2})((\mathsf{FAI}|\mathsf{KIL}), 1)(x_{3,1}, x_{3,2})(x_{4,1}, 1)$ while V^{a} remains unchanged again. Recall that we identify single values with singleton intervals.

Suppose that the algorithm selects $x_{1,2}$ next. Again the algorithm calls Function 3 with $\varrho = \nabla(\mathsf{priority}) = [1, 2, 8, 9, 10]$ in Line 20, this time returning the query $q = (s, w, c)$ with $(x_{1,1}, [1, 2])((\mathsf{FAI}|\mathsf{KIL}), 1)(x_{3,1}, x_{3,2})(x_{4,1}, 1)$ and True.

Assume that the next variable chosen by Algorithm 1 is $x_{1,1}$. Since $A_1 = \mathsf{task}$ and $1 \in U$, the algorithm enters the for-loop in Line 11 and calls Function 2 first with Δ^1 (note that V^{a} is still empty), before the function is called again with $\Omega = \Delta^2 = \Delta^2_{\mathsf{task}} = \{\{\mathsf{SUB}, \mathsf{UPD}\}, \{\mathsf{SUB}, \mathsf{FAI}\}, \{\mathsf{SCH}, \mathsf{UPD}\}, \{\mathsf{SCH}, \mathsf{FAI}\}, \{\mathsf{UPD}, \mathsf{KIL}\}, \{\mathsf{KIL}, \mathsf{FAI}\}\}$. It turns out that none of these disjunctions lead to an admissible replacement operation. Hence, the function returns an unchanged query as well as False. No further iteration of the for-loop is performed because d is bounded by 2. Overall, no replacement operation was possible, hence the variable $x_{1,1}$ remains in the query string and gets available in Line 22, i. e. $V^{\mathrm{a}} = \{x_{1,1}\}$.

We assume that during the sixth and seventh transition through the main loop the algorithm selects the unvisited variables $x_{3,2}$ and $x_{4,1}$. It can be easily verified that the following replacements are admissible: $s\langle x_{3,2} \mapsto [9, 10]\rangle$ and $s\langle x_{4,1} \mapsto \{\mathsf{FIN}\}\rangle$, leading to the query string $(x_{1,1}, [1, 2])((\mathsf{FAI}|\mathsf{KIL}), 1)(x_{3,1}, [9, 10])(\mathsf{FIN}, 1)$.

In its last iteration (during the first iteration of the for-loop in Line 11 and the corresponding call of Function 2 in Line 13), the only admissible replacement operation is $s\langle x_{3,1} \mapsto x_{1,1}\rangle$. The run terminates after this iteration and outputs the query $q = (s, w, c)$ with $(x_{1,1}, [1, 2])((\mathsf{FAI}|\mathsf{KIL}), 1)(x_{1,1}, [9, 10])(\mathsf{FIN}, 1)$. ⌟

Consider a scenario in which the variable $x_{m,n}$ can not be replaced and $m \in O$. This implies, that the only interval leading to a query that satisfies the support threshold would be $[\min(\varrho), \max(\varrho)]$, for $\varrho := \nabla(n)$, which is equivalent to a variable. If a variable is already

available, it makes sense to check for a replacement by this variable before searching for an admissible replacement by an interval in order to prevent the logarithmic search.

As illustrated by the example above, the algorithm produces a sequence $s_0, s_1, \ldots, s_{k\cdot\ell}$ of query strings such that for every $i \in [k \cdot \ell]$, either s_i was obtained from s_{i-1} by a replacement operation, or $s_i = s_{i-1}$. Furthermore, we note that $q_0 \xrightarrow{\text{hom}} q_1 \xrightarrow{\text{hom}} \ldots \xrightarrow{\text{hom}} q_{k\cdot\ell}$, whereby $q_i = (s_i, w, c)$ is the (k, ℓ, w, c, d)-query with $s = s_i$ for every $i \in [k \cdot \ell]$. By Theorem 16, this implies $\mathsf{Mod}(q_0) \supseteq \mathsf{Mod}(q_1) \supseteq \ldots \supseteq \mathsf{Mod}(q_{k\cdot\ell})$. As we will formalise next, it can be guaranteed that the final query $q_{k\cdot\ell}$ is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}$, sp, and (k, ℓ, w, c, d). In order to prove this we need replacement operations to be defined in the correct way (i. e. as by Algorithm 1).

Note that the algorithm presented in this paper can be used to discover descriptive mswg-queries and subsequence-queries with generalised gap-size constraints as well, by restricting the schema to a shared unordered domain and changing the local gap-size constraints to generalised gap-size constraints in the sense of [8]. We would like to stress the point that due to the arbitrary choices in Line 6 of Algorithm 1 and Line 2 of Function 2, different runs of the presented discovery algorithm may lead to different queries. Note that due to the same reasons as discussed in [12] for the base case of swg-queries, not all queries can be computed by the presented algorithm. We refer to [8] for a discussion on how disjunctions can be calculated in case that $d > 1$. The following theorem states that any choice is fine, and that any output query is guaranteed to be a descriptive query.

Theorem 25. *Let $k \in \mathbb{N}_{\geqslant 1}$ and let $\mathcal{A} = (A_1, \ldots, A_k)$ be an event schema with $|dom(A_i)| \geqslant 2$ for all $i \in [k]$. Let $\mathcal{S}$ be a k-dimensional sample over $\mathcal{A}$, let sp be a support threshold with $0 < \mathsf{sp} \leqslant 1$, let (k, ℓ, w, c, d) be query parameters.*

(a) If there does not exist any (k, ℓ, w, c, d)-dswg$^{\leqslant}$-query that is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}$, sp, and (k, ℓ, w, c, d) then there is only one run of Algorithm 1 upon the defined input, and it stops in Line 2 with output $\perp$.

(b) Otherwise, every run of Algorithm 1 upon input $(\mathcal{S}, \mathsf{sp}, (k, \ell, w, c, d))$ terminates and outputs a dswg$^{\leqslant}$-query q with $|\chi| \leqslant d$ for all $\chi \in \mathbb{D}(q)$, that is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}$, sp, and (k, ℓ, w, c, d).

For the special case of $k = d = 1$ over a schema sharing one unordered domain, an analogous theorem was proved in [12]. It was then extended in [13] to the multi-dimensional scenario (over a schema sharing only one unordered domain over all attributes) and adapted by Frochaux et al. in [8] to allow for disjunctive queries and queries with generalised gap-size constraints (in the one-dimensional setting). In this paper, we lift the previous version of this theorem for disjunctive subsequence-queries to the multi-dimensional setting and extend it by intervals, thereby reducing the gap between subsequence-queries and other query models in CEP.

Proof of (a). Consider the case where there does not exist any (k, ℓ, w, c, d)-dswg$^{\leqslant}$-query

that is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}$, sp, and (k, ℓ, w, c, d). This implies $\mathsf{supp}(q_{mg}, \mathcal{S}) < \mathsf{sp}$ and every query q' with $\mathsf{Mod}(q') \subseteq \mathsf{Mod}(q_{mg})$ satisfies $\mathsf{supp}(q', \mathcal{S}) \leqslant \mathsf{supp}(q_{mg}, \mathcal{S}) < \mathsf{sp}$ as well. In this case, the algorithm stops in Line 2 and outputs an error message $\perp$. □

The proof of (b) is based on the following idea: Let $\hat{r} \coloneqq k \cdot \ell = |\mathbb{V}(s_{mg})|$. First note, that each output query $q' := q_{\hat{r}}$ is a (k, ℓ, w, c, d)-query with $\mathsf{supp}(q_{\hat{r}}, \mathcal{S}) \geqslant \mathsf{sp}$. For contradiction, let us assume that $q_{\hat{r}}$ is not descriptive. Then there exists a (k, ℓ, w, c, d)-query $\tilde{q}$ that satisfies the support threshold as well and that is strictly more restrictive than $q_{\hat{r}}$, i. e. $\mathsf{Mod}(\tilde{q}) \subsetneq \mathsf{Mod}(q_{\hat{r}})$ and therefore, $q_{\hat{r}} \xrightarrow{\text{hom}} \tilde{q}$ and $\tilde{q} \not\equiv q_{\hat{r}}$ (see Remark 23). The goal is to prove that for every $i \in \{0, \ldots, \hat{r}\}$, the queries q_i and $\tilde{q}$ are partially isomorphic w.r.t. the positions of the variables already considered by the algorithm. This leads to the fact that $\tilde{q} \cong q_{\hat{r}}$ for $i = \hat{r}$, which contradicts our assumption.

We refer to Appendix B for the full proof of the second statement of the Theorem. Note that the main differences in the proof (compared to previous versions) lay (1) in the extension to the multi-dimensional scenario for disjunctions, and (2) in the notion of a schema over ordered and unordered domains, and therefore, in the correctness of InterOp and Function 3.

Regarding the complexity of Algorithm 1, we identify two bottle necks. The first is already known from [12, 8] and is caused by the recurring calls of a *matching* subroutine. The referred results imply NP-hardness for our algorithm as we can use it as well for swg-queries from [12]. Membership can be obtained by guessing a witness. The second bottle-neck is the Δ-calculation in the case of $d \geqslant 2$, which may lead to an exponential blow-up regarding the number of calls to the matching subroutine. This can be handled by adjusting the parameter d, i. e. by bounding the size of the disjunctive clauses in the query string. Note that compared to the exponential blow-up for disjunctions, we only need a logarithmic search of an interval over an ordered domain. For ordered domains, we therefore resolved the second bottle-neck.

Lastly we would like to point out that there are multiple ways of discovering intervals, which again may be beneficial or unfavourable depending on the underlying domain. If we for example consider a scenario in which we are looking for high temperature values in order to detect a fire, one may strive for an algorithm that prefers intervals with higher values over intervals with lower values. Another optimisation approach could be to minimise the difference of the upper and lower bound of an interval, in order to find a "global" optimum in contrast to the local (but still descriptive) optimum Function 3 is able to compute. One way to achieve this could be by pursuing all possible intervals considered in Function 3. Note that this would increase the number of calls to the matching subroutine. An obvious approach for optimisation would be to randomise the order in which the bounds on the intervals are tightened, which may reduce the bias to the centre of the active domain. We defer an in-depth discussion on algorithms for discovering intervals to future work.

5 Final Remarks

Motivated by query languages in the field of Complex Event Processing that allow for different kinds of inequalities and previous work on the discovery of multi-dimensional or disjunctive subsequence-queries with wildcards and local gap-size constraints, we defined a query model that allows for disjunction over values and intervals over ordered domains, i. e. lower and upper bounds on values that shall occur in a multi-dimensional trace (Section 3). Furthermore, we discussed a suitable characterisation of query containment and isomorphic queries (Section 3.2). We defined the task of query discovery for mdswg$^{\leqslant}$-queries and extended previous discovery algorithms for multi-dimensional or disjunctive subsequence-queries to our new query model, in order to solve the query discovery problem (Section 4).

As future work, we plan to extend the prototypical implementation of query discovery that exists for earlier subsequence-query models and perform a comprehensive experimental analysis. Here, the focus will be on the discovery of intervals: as discussed at the end of Section 4, multiple algorithms can be used to compute intervals. We are interested in solving the question for which sample characteristics which approaches perform well.

As a second line of research we will investigate the connection between mdswg$^{\leqslant}$-queries and Conjunctive Queries with Inequalities. Note that efficient query evaluation (deciding whether a given query is satisfied on a given database) and query enumeration (output all tuples of a given database that satisfy the given query) are central problems in all database related fields. In various real-world applications, the given database is changing over time. Note that event streams may be considered as a database that is initially empty and gets insertion updates whenever a new event is detected. Furthermore, correlation among events in CEP can be expressed by means of CQs with inequalities in order to compare timestamps, as shown in the following example.

Example 26. Consider again the traces and query q from Example 1. We extend these with a new first attribute recording timestamps. Then a trace can be seen as a relational database, while the "PATTERN SEQ"- and "WHERE"-clause of the CEP query q can be seen as the following conjunctive query on such databases:

$$\begin{aligned} Ans() \leftarrow\ & R(\tau_1, x, 1),\ R(\tau_2, \mathsf{FAI}, 1),\ R(\tau_3, x, 9),\ R(\tau_4, \mathsf{FIN}, 1), \\ & \tau_1 < \tau_2,\ \tau_2 < \tau_3,\ \tau_3 < \tau_4 \end{aligned}$$

where $\tau_1, \ldots, \tau_4$ and x are variables capturing the timestamps and tasks of events. Intervals as introduced in Example 3 could be added to the query by appending $1 \leqslant x_{p_1} \leqslant 2,\ 9 \leqslant x_{p_4} \leqslant 10$. Note that the query does not capture the WITHIN-clause of q. ⌟

Given the correspondence between CEP and CQs with inequalities as well as event streams and databases under updates, it might be promising to investigate whether we can transfer results developed from one field to the other. We would like to stress the point that the investigation of mdswg$^{\leqslant}$-queries enables this investigation in future work. Hence, they are laying a foundation for knowledge transfer between both worlds: CEP and CQ.

References

[1] Dana Angluin. “Inductive Inference of Formal Languages from Positive Data”. In: *Inf. Control.* 45.2 (1980), pp. 117–135. DOI: 10.1016/S0019-9958(80)90285-5.

[2] 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*. ACM, 2017, pp. 7–10. DOI: 10.1145/3093742.3095106.

[3] Alexander Artikis et al. “Heterogeneous Stream Processing and Crowdsourcing for Urban Traffic Management”. In: *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*. OpenProceedings.org, 2014, pp. 712–723. DOI: 10.5441/002/edbt.2014.77.

[4] Gianpaolo Cugola and Alessandro Margara. “Processing flows of information: From data stream to complex event processing”. In: *ACM Comput. Surv.* 44.3 (2012), 15:1–15:62. DOI: 10.1145/2187671.2187677.

[5] 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)*. 2021, 25:1–25:19. DOI: 10.4230/LIPIcs.STACS.2021.25.

[6] Henning Fernau, Florin Manea, Robert Mercas, and Markus L. Schmid. “Revisiting Shinohara’s algorithm for computing descriptive patterns”. In: *Theor. Comput. Sci.* 733 (2018), pp. 44–54. DOI: 10.1016/j.tcs.2018.04.035.

[7] André Frochaux and Sarah Kleest-Meißner. “Puzzling over Subsequence-Query Extensions: Disjunction and Generalised Gaps”. In: *CoRR* 2305.08236 (2023), pp. 1–22. DOI: 10.48550/arXiv.2305.08236. arXiv: 2305.08236.

[8] André Frochaux 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*. Ed. by Benny Kimelfeld, Maria Vanina Martinez, and Renzo Angles. Vol. 3409. CEUR Workshop Proceedings. CEUR-WS.org, 2023, pp. 1–12. URL: https://ceur-ws.org/Vol-3409/paper3.pdf.

[9] 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)*. 2021, 34:1–34:18. DOI: 10.4230/LIPIcs.STACS.2021.34.

[10] Lars George, Bruno Cadonna, and Matthias Weidlich. “IL-Miner: Instance-Level Discovery of Complex Event Patterns”. In: *Proc. VLDB Endow.* 10.1 (2016), pp. 25–36. DOI: 10.14778/3015270.3015273.

[11] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. “Complex event recognition in the Big Data era: a survey”. In: *VLDB J.* 29.1 (2020), pp. 313–352. DOI: 10.1007/s00778-019-00557-w.

[12] 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*. Vol. 220. LIPIcs. 2022, 18:1–18:21. DOI: 10.4230/LIPIcs.ICDT.2022.18.

[13] 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*. Ed. by Birgitta König-Ries, Stefanie Scherzinger, Wolfgang Lehner, and Gottfried Vossen. Vol. P-331. LNI. Gesellschaft für Informatik e.V., 2023, pp. 511–533. DOI: 10.18420/BTW2023-24.

[14] 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*. 2019, pp. 1–27. DOI: 10.1007/978-3-030-28796-2_1.

[15] Alessandro Margara, Gianpaolo Cugola, and Giordano Tamburrelli. “Learning from the past: automated rule generation for complex event processing”. In: *The 8th ACM International Conference on Distributed Event-Based Systems, DEBS ’14, Mumbai, India, May 26-29, 2014*. ACM, 2014, pp. 47–58. DOI: 10.1145/2611286.2611289.

[16] Grzegorz Rozenberg and Arto Salomaa. “Patterns”. In: *Handbook of Formal Languages*. Vol. 1. Springer, 1997, pp. 230–242.

[17] Takeshi Shinohara and Setsuo Arikawa. “Pattern Inference”. In: *Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report*. 1995, pp. 259–291. DOI: 10.1007/3-540-60217-8_13.

[18] Kia Teymourian, Malte Rohde, and Adrian Paschke. “Knowledge-based processing of complex stock market events”. In: *15th International Conference on Extending Database Technology, EDBT ’12, Berlin, Germany, March 27-30, 2012, Proceedings*. ACM, 2012, pp. 594–597. DOI: 10.1145/2247596.2247674.

[19] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. “Large-scale cluster management at Google with Borg”. In: *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21-24, 2015*. ACM, 2015, 18:1–18:17. DOI: 10.1145/2741948.2741964.

[20] 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*. Ed. by Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu. ACM, 2014, pp. 217–228. DOI: 10.1145/2588555.2593671.

Appendix

A Details Omitted in Section 3

A.1 Proof of the first claim of Theorem 16

Let t be of the form $e_1 \dots e_n$, let h be a homomorphism witnessing $q' \xrightarrow{\text{hom}} q$ and let (μ, ξ) be a witness for $t \models q$, i. e., $t \in \mathsf{Mod}(q)$. Let μ' be defined by $\mu'((i,j),z) \coloneqq \mu((i,j), h((i,j),z))$ for all $(i,j) \in [\ell]\times[k]$ and all $z \in (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ Since q and q' have the same window size w and local gap-size constraints c, it suffices to show that (μ', ξ) is a witness for $t \models q'$. And since $\mu'((i,j), s'[i][j]) = \mu((i,j), h((i,j), s'[i][j])) = \mu((i,j), s[i][j]) \ni t[\xi(i)][j]$ by definition, it only remains to show that μ' is a valid final substitution. Let $(i,j) \in [\ell]\times[k]$ and let $z \in (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$. We distinguish the following cases.

Case 1: $s'[i][j]$ is of the form $[p,q] \in \mathbb{I}_j$. Since h is a homomorphism (and thus also a substitution), $h(s'[i][j])$ is of the form $[p',q']$ where $[p',q'] \subseteq [p,q]$. Because μ is final, we have $\mu([p',q']) = [\gamma,\gamma]$ with $\gamma \in [p',q']$. In total, we get that $\mu'((i,j),[p,q]) = [\gamma,\gamma]$ with $[\gamma,\gamma] \subseteq [p,q]$.

Case 2: $s'[i][j]$ is of the form $\chi \in \mathbb{D}_j$. By definition, $h(s'[i][j])$ is of the form χ' where $\chi' \subseteq \chi$. Since μ is final, we have $\mu((i,j),\chi') = \{\gamma\}$ where $\{\gamma\} \subseteq \chi'$. In total, we get that $\mu'((i,j),\chi) = \{\gamma\}$ with $\{\gamma\} \subseteq \chi$.

Case 3: $s'[i][j]$ is of the form $x \in \mathbb{V}$. First, notice that since h is a substitution, there is a function f_h such that all occurrences of x in s' are mapped to $f_h(x)$. We have to ensure that μ' maps every occurrence of x onto the same singleton set or interval. We distinguish the following cases based on where h maps x.

(a): If $h((i,j),x) = y \in \mathbb{V}$, then by definition, $\mu'((i,j),x) = \mu((i,j),h(x)) = \mu((i,j),y)$ and since μ is a substitution, there is a function $f_\mu\colon \mathbb{V} \to (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ such that $\mu((i,j),y) = f_\mu(y)$. Since μ is a final substitution, $f_\mu(y)$ is a singleton set or a singleton interval, depending on whether $dom(A_j)$ is ordered or not. In both cases, $\mu'((i,j),x)$ maps to a singleton set or singleton interval, and since all occurrences of x in s' are mapped to y, μ' is a valid final substitution.

(b): If $h((i,j),x) \notin \mathbb{V}$ and $dom(A_j)$ is ordered, then $h(x) = [p,q] \in \mathbb{I}_j$. If x does not appear multiple times in s', then by similar reasoning as in the first case, $\mu'((i,j),x) = [\gamma,\gamma]$ for some $\gamma \in [p,q]$, and thus, μ' is a valid final substitution.

Should x appear multiple times, then the additional requirement of homomorphisms tells us that h maps each occurrence of x onto the same singleton interval $[\gamma, \gamma]$ for some $\gamma \in dom(A_j)$. Since it is singleton, μ (and therefore also μ') map $[\gamma, \gamma]$ to itself, i. e., every occurrence of x in s' is mapped to $[\gamma, \gamma]$. Hence, μ' is a valid final substitution.

(c): If $h((i, j), x) \notin \mathbb{V}$ and $dom(A_j)$ is unordered, then $h(x) = \chi \in \mathbb{D}_j$. If x does not appear multiple times in s', then by similar reasoning as in the second case, $\mu'((i, j), x) = \{\gamma\}$ for some $\gamma \in \chi$, and thus, μ' is a valid final substitution.
Should x appear multiple times, then the additional requirement of homomorphisms tells us that h maps each occurrence of x onto the same singleton set $\{\gamma\}$ for some $\gamma \in dom(A_j)$. Since it is singleton, μ (and therefore also μ') map $\{\gamma\}$ to itself, i. e., every occurrence of x in s' is mapped to $\{\gamma\}$. Hence, μ' is a valid final substitution. ■

A.2 Proof of Lemma 19

Proof. The equivalence "$q \xrightarrow{\text{hom}} q'$ and $q' \xrightarrow{\text{hom}} q \iff \mathsf{Mod}(q) = \mathsf{Mod}(q')$" is a direct consequence of Theorem 16.

First, let $q \cong q'$. Then there is a bijection $\pi\colon (\mathbb{V}(q) \cup \mathbb{I} \cup \mathbb{D}) \to (\mathbb{V}(q') \cup \mathbb{I} \cup \mathbb{D})$, such that $\pi(s[i][j]) = s'[i][j]$ for all $i \in [\ell]$, $j \in [k]$ and $\pi_{|_{(\mathbb{I} \cup \mathbb{D})}} = \mathrm{id}$, i. e., $\pi(z) = z$ for all $z \in (\mathbb{I} \cup \mathbb{D})$.

Hence, by definition, $h\colon \big(([\ell]\times[k]) \cup (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})\big) \to (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ defined by

$$h((i, j), z) \coloneqq \begin{cases} \pi(z) & \text{if } z \in \mathbb{V}(q), \\ y & \text{if } z \in \mathbb{V} \setminus \mathbb{V}(q) \text{ where } y \in \mathbb{V} \setminus (\mathbb{V}(q) \cup \mathbb{V}(q')), \\ z & \text{if } z \in (\mathbb{I} \cup \mathbb{D}), \end{cases}$$

is a homomorphism from q to q'. Conversely, since π is bijective and $\pi^{-1}_{|_{(\mathbb{I} \cup \mathbb{D})}} = \mathrm{id}$, $h'\colon \big(([\ell]\times[k]) \cup (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})\big) \to (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ defined by

$$h'((i, j), z) \coloneqq \begin{cases} \pi^{-1}(z) & \text{if } z \in \mathbb{V}(q'), \\ y & \text{if } z \in \mathbb{V} \setminus \mathbb{V}(q') \text{ where } y \in \mathbb{V} \setminus (\mathbb{V}(q) \cup \mathbb{V}(q')), \\ z & \text{if } z \in (\mathbb{I} \cup \mathbb{D}), \end{cases}$$

is a homomorphism from q' to q. This shows the forward direction "⇒".

For the backward direction "⇐", we prove that $q \xrightarrow{\text{hom}} q'$ and $q' \xrightarrow{\text{hom}} q$ implies $q \cong q'$. Let h and h' be homomorphisms from q to q' and from q' to q, respectively. We show that π defined by $\pi(z) \coloneqq f_h(z)$ if $z \in \mathbb{V}(q)$, and $\pi(z) \coloneqq z$ if $z \in (\mathbb{I} \cup \mathbb{D})$ is an isomorphism from q to q' and therefore witnesses $q \cong q'$. By definition of h and h', it holds that $h((i, j), s[i][j]) = s'[i][j]$ and $h'((i, j), s'[i][j]) = s[i][j]$ for all $(i, j) \in [\ell]\times[k]$, and

thus, $h((i,j),z) = h'((i,j),z) = z$ for all $z \in (\mathbb{I}(q) \cup \mathbb{I}(q') \cup \mathbb{D}(q) \cup \mathbb{D}(q'))$. This already shows that $\pi(s[i][j]) = s'[i][j]$ for all $(i,j) \in [\ell]\times[k]$ where $s[i][j] \notin \mathbb{V}$ or $s'[i][j] \notin \mathbb{V}$.

It remains to consider the variables. Note that $s[i][j] \in \mathbb{V} \iff s'[i][j] \in \mathbb{V}$ due to the reasoning above. Further, it is easy to see that π is surjective on $\mathbb{V}(q')$. Thus, it remains to verify that π is injective on $\mathbb{V}(q)$. Let $x, y \in \mathbb{V}(q)$ such that $\pi(x) = f_h(x) = z = f_h(y) = \pi(y)$. Then there are $(i,j) \neq (\hat{\imath},\hat{\jmath}) \in [\ell]\times[k]$ such that $s[i][j] = x$, $s[\hat{\imath}][\hat{\jmath}] = y$ and $s'[i][j] = z$, $s'[\hat{\imath}][\hat{\jmath}] = z$. Since h' is a substitution, $h'((i,j),z) = f_{h'}(z) = h'((\hat{\imath},\hat{\jmath}),z)$ holds, as well as $h'((i,j),z) = s[i][j]$, $h'((\hat{\imath},\hat{\jmath}),z) = s[\hat{\imath}][\hat{\jmath}]$. Hence, $s[i][j] = s[\hat{\imath}][\hat{\jmath}]$, i. e., π is injective on $\mathbb{V}(q)$.

It is now easy to see that π can be extended (onto all the variables not present in $\mathbb{V}(q)$) to be bijective on $(\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ such that $\pi_{|_{(\mathbb{I}\cup\mathbb{D})}} =$ id and for all $(i,j) \in [\ell]\times[k]$ it holds that $\pi(s[i][j]) = s'[i][j]$. □

A.3 Proof of Lemma 20

Proof. To prove the backward direction, we assume that $q \cong q'$. This means, there exists a $\pi\colon (\mathbb{V}(q) \cup \mathbb{I} \cup \mathbb{D}) \to (\mathbb{V}(q') \cup \mathbb{I} \cup \mathbb{D})$, such that $\pi_{|_{(\mathbb{I}\cup\mathbb{D})}} =$ id and $\pi(s[i][j]) = s'[i][j]$ for all $(i,j) \in [\ell]\times[k]$. Since we have, that $\pi_{|_{(\mathbb{I}\cup\mathbb{D})}} =$ id, it holds for all $(i,j) \in [\ell] \times [k]$ with $s[i][j] \in (\mathbb{I} \cup \mathbb{D})$, that $s[i][j] = \pi(s[i][j]) = s'[i][j] \in (\mathbb{I} \cup \mathbb{D})$. This implies the following two facts.

- $s[i][j] \in (\mathbb{I} \cup \mathbb{D}) \iff s'[i][j] \in (\mathbb{I} \cup \mathbb{D})$ and
- $s'[i][j] = s'[i'][j']$ for all $(i,j),(i',j) \in [\ell] \times [k]$ where $s[i][j] = s[i'][j'] \in (\mathbb{I} \cup \mathbb{D})$.

The first fact implies for all $(i,j) \in [\ell] \times [k]$ that $s[i][j] \in \mathbb{V}(q) \iff s'[i][j] \in \mathbb{V}(q')$. Together with $\pi(s[i][j]) = s'[i][j]$ we have: $s[i][j] = s[i'][j'] \Rightarrow s'[i][j] = s'[i'][j']$ for all $(i,j),(i',j') \in I$. Since π is injective, $s'[i][j] = \pi(s[i][j]) = \pi(s[i'][j']) = s'[i'][j']$ in turn implies $s[i][j] = s[i'][j']$.

For the forward direction, we have $q \sim_{[\ell]\times[k]} q'$ by assumption. For every $x \in \mathbb{V}(q)$, we define $i_x \coloneqq \min\{\, i \in [\ell] \;:\; \text{there exists } j \in [k], \text{s.t. s[i][j]=x} \,\}$ and $j_x \coloneqq \min\{\, j \in [k] \;:\; s[i_x][j] = x \,\}$. Intuitively, $s[i_x][j_x]$ is the first occurrence of variable x in the query string s. We define $\pi\colon (\mathbb{V}(q) \cup \mathbb{I} \cup \mathbb{D}) \to (\mathbb{V}(q') \cup \mathbb{I} \cup \mathbb{D})$ via $\pi(z) \coloneqq z$ for all $z \in (\mathbb{D} \cup \mathbb{I})$ and $\pi(x) \coloneqq s'[i_x][j_x]$ for all $x \in \mathbb{V}(q)$.

First we ensure, that π is bijective. Consider $x, y \in \mathbb{V}(q)$ with $\pi(x) = \pi(y)$, therefore we have $s'[i_x][j_x] = s'[i_y][j_y]$. By the first property of partial isomorphisms, namely $s[i][j] = s[i'][j'] \iff s'[i][j] = s'[i'][j']$ for all $(i,j),(i',j') \in [\ell] \times [k]$, this implies $s[i_x][j_x] = s[i_y][j_y]$, i.e. $x = y$. This means, π is injective.
Now, consider an arbitrary $y \in \mathbb{V}(q')$. Let $i'_y \coloneqq \min\{\, i \in [\ell] \;:\; \text{there exists } j \in [k], \text{s.t. s'[i][j]=y} \,\}$ and $j'_y \coloneqq \min\{\, j \in [k] \;:\; s'[i'_y][j] = y \,\}$. Now

$s'[i'_y][j'_y]$ denotes the first occurrence of variable y in the query string s'. Let $x \coloneqq s[i'_y][j'_y]$. By the first property of partial isomorphisms, we obtain with $x = s[i'_y][j'_y] = s[i_x][j_x]$ that $s'[i'_y][j'_y] = s'[i_x][j_x]$. Hence, $y = s'[i'_y][j'_y] = s'[i_x][j_x] = \pi(x)$ and therefore π is surjective. In summary, $\pi\colon (\mathbb{V}(q) \cup \mathbb{I} \cup \mathbb{D}) \to (\mathbb{V}(q') \cup \mathbb{I} \cup \mathbb{D})$ is a bijection with $\pi(z) \coloneqq z$ for all $z \in (\mathbb{D} \cup \mathbb{I})$.

To end this proof, we have to ensure, that $\pi(s[i][j]) = s'[i][j]$ for all $(i, j) \in [\ell] \times [k]$. For any $(i, j) \in [\ell] \times [k]$ with $s[i][j] \in (\mathbb{I} \cup \mathbb{D})$, we know by the second property of partial isomorphisms, that $s'[i][j] \in (\mathbb{I} \cup \mathbb{D})$ and $s[i][j] = s'[i][j]$, then by definition of π it holds that $s[i][j] = \pi(s[i][j]) = s'[i][j]$. Finally, for any $(i, j) \in [\ell] \times [k]$ with $s[i][j] = x \in \mathbb{V}(q)$, by the first property of partial isomorphisms and $s[i][j] = s[i_x][j_x]$, we obtain $s'[i][j] = s'[i_x][j_x]$ and therefore $\pi(s[i][j]) = \pi(x) = s'[i_x][j_x] = s'[i][j]$. □

B Details Omitted in Section 4

Recall that given a (k, ℓ, w, c, d)-query $q = (s, w, c)$, a variable $x \in \mathbb{V}(q)$ and a symbol $z \in (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ we write $s\langle x \mapsto z\rangle$ to denote the query string s' which is obtained from s by replacing every occurrence of x in s by z. Furthermore, we let $pos(q, z) = pos(s, z) = \{(i, j) \in \big([\ell] \times [k]\big) \,:\, s[i][j] = z\}$ be the set of all positions i in s that carry z. Given a set of positions $P \subseteq [\ell] \times [k]$, and a symbol $z \in (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ we write $s\langle P \mapsto z\rangle$ to denote the query string s' which is obtained from s by setting $s[i][j]$ to z, for all $(i, j) \in P$.

Theorem 25. *Let $k \in \mathbb{N}_{\geqslant 1}$ and let $\mathcal{A} = (A_1, \ldots, A_k)$ be an event schema with $|dom(A_i)| \geqslant 2$ for all $i \in [k]$. Let $\mathcal{S}$ be a k-dimensional sample over $\mathcal{A}$, let* sp *be a support threshold with $0 <$ sp $\leqslant 1$, let (k, ℓ, w, c, d) be query parameters.*

(a) If there does not exist any (k, ℓ, w, c, d)-dswg$^{\leqslant}$-query that is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}$, sp*, and (k, ℓ, w, c, d) then there is only one run of Algorithm 1 upon the defined input, and it stops in Line 2 with output $\perp$.*

(b) Otherwise, every run of Algorithm 1 upon input $(\mathcal{S},$ sp*$, (k, \ell, w, c, d))$ terminates and outputs an dswg$^{\leqslant}$-query q with $|\chi| \leqslant d$ for all $\chi \in \mathbb{D}(q)$, that is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}$,* sp*, and (k, ℓ, w, c, d).*

Note that during each iteration of the main loop of Algorithm 1, the algorithm either replaces the current variable by an interval, a disjunction, an available variable or does not change the query string at all. We call these operations InterOp, DisjOp, VarOp, or NoChangeOp and say that q_r is obtained by q_{r-1} by performing either a replacement operation (either InterOp, DisjOp, or VarOp) or NoChangeOp, for all $r \geqslant 1$.

Throughout the proof, we heavily make use of Remark 23 that holds for mdswg$^{\leqslant}$-queries if $|dom(A_j)| \geqslant 2$ for all $j \in [k]$. We will refer to Algorithm 1 also as the *main algorithm*.

Proof of (b). Assume $\mathsf{supp}(q_{mg}, \mathcal{S}) \geqslant \mathsf{sp}$, for the most general k-dswg$^{\leqslant}$-query defined in Line 1. In this case, every run of the main algorithm reaches Line 6 and proceed from there on.

Let Δ be the set of disjunctions defined in Line 3. Note that every query q' that is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}$, sp, and (k, ℓ, w, c, d), satisfies $\mathbb{D}(q') \subseteq \Delta$, because otherwise, $\mathsf{supp}(q', \mathcal{S})$ would be $< \mathsf{sp}$. Particularly, it holds for each query q that is computed by Algorithm 1 that $\mathbb{D}(q) \subseteq \Delta$ and $|\chi| \leqslant d$, for all $\chi \in \Delta$, due to Line 3.

Let us first argue that every iteration r of the outer while-loop starting in Line 6, will end. First note that the set of available disjunctions and variables $\Delta \cup V^{\mathrm{a}}$ is finite, since it is bounded by the number of values occurring in the given sample at an attribute index $i \in U$ and the number of variables in s_{mg}, which equals $k \cdot \ell$. During the call of Function 2 in Line 17 the while-loop starting in Line 1 of the function ends after at most $\Omega = |V^{\mathrm{a}}|$ iterations and during each iteration the current variable is either replaced by a variable (VarOp), or remains in the query string if no replacement operation is possible. Each call to Function 3 will end as well since during each iteration of the while-loop starting in Line 4, the given interval is decreased. This can be done only logarithmically often in the worst case until an interval consisting of only one value remains. During the i-th (of a bounded number of d) iterations through the for-loop (Line 11), the while-loop in Function 2 starting in Line 1 ends after at most $\Omega = |\Delta^i| = |\bigcup_{n \in U} \Delta^i_{A_n}|$ iterations and during each iteration the current variable is either replaced by a disjunction (DisjOp) or variable (VarOp), or remains in the query string if no replacement operation is possible (NoChangeOp).

Let us now fix a particular run of Algorithm 1. Let $q_0 = q_{mg}$, $s_0 = s_{mg}$, $V^{\mathrm{u}}_0 \coloneqq \mathbb{V}(q_0)$ and $V^{\mathrm{a}}_0 \coloneqq \varnothing$. For every $r \in \{1, 2, \dots\}$ let s_r, V^{u}_r, V^{a}_r be the query string s and the sets V^{u} and V^{a} at the end of the r-th iteration through the outer while-loop of the main algorithm, and let q_r be the query (s_r, w, c). Furthermore, for each $r \geqslant 1$ let x_r be the particular element in V^{u} that is chosen at the beginning of the r-th iteration through the outer while-loop of the main algorithm.

By induction on r and by construction of the algorithm, it is straightforward to prove the following claim. The proof is provided at the end of this proof.

Claim 28. *For every $r \geqslant 1$ we have*

1. $V^{\mathrm{u}}_r = \mathbb{V}(q_0) \setminus \{x_1, \dots, x_r\}$ *and* $V^{\mathrm{u}}_r \cap V^{\mathrm{a}}_r = \varnothing$ *and* $V^{\mathrm{u}}_r \cup V^{\mathrm{a}}_r = \mathbb{V}(s_r)$.
2. $\mathbb{D}(q_r) \subseteq \Delta$ *and* $\mathsf{supp}(q_r, \mathcal{S}) \geqslant \mathsf{sp}$.
3. $q_{r-1} \xrightarrow{\text{hom}} q_r$.
4. *For every* $x \in V^{\mathrm{u}}_r$ *we have* $pos(s_r, x) = pos(s_0, x)$.
5. $s_r[i][v] = s_{r-1}[i][v]$ *for all* $(i, v) \in \big([\ell] \times [k]\big) \setminus pos(s_0, x_r)$.

From this claim we obtain that after $\hat{r} := |\mathbb{V}(q_0)| = k \cdot \ell$ iterations through the outer while-loop of the main algorithm, its run terminates with $V^{\mathrm{u}}_{\hat{r}} = \varnothing$ and outputs an (k, ℓ, w, c, d)-query $q_{\hat{r}}$ with $\mathsf{supp}(q_{\hat{r}}, \mathcal{S}) \geqslant \mathsf{sp}$ and $q_0 \xrightarrow{\text{hom}} q_{\hat{r}}$ (i. e., by Theorem 16, $\mathsf{Mod}(q_{\hat{r}}) \subseteq \mathsf{Mod}(q_0)$).

We need to show that this query $q_{\hat{r}}$ is descriptive for $\mathcal{S}$ w.r.t. $\mathcal{A}$, sp, and (k, ℓ, w, c, d). For contradiction, assume that it is not. Then, according to Remark 23, there exists a (k, ℓ, w, c, d)-query q' with $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$, $q_{\hat{r}} \xrightarrow{\text{hom}} q'$ and $q' \not\equiv q_{\hat{r}}$.

From Claim 28(3) we know that $q_r \xrightarrow{\text{hom}} q_{\hat{r}}$ for all $r \leqslant \hat{r}$. Due to $q_{\hat{r}} \xrightarrow{\text{hom}} q'$, it holds that

$$q_r \xrightarrow{\text{hom}} q' \quad \text{for all } r \in \{0, 1, \ldots, \hat{r}\}. \tag{$\dagger$}$$

Let s' be the query string of q'. In order to deduce the desired contradiction, the notion of partially isomorphic queries will be crucial. For each $r \in \{0, 1, \ldots, \hat{r}\}$ let

$$\rho_r \;\coloneqq\; \{\, (i,u) \in [\ell] \times [k] \;:\; s_r[i][u] \in \Delta \cup \mathbb{I} \,\} \;\cup\; \bigcup_{\nu=1}^{r} pos(s_0, x_\nu)\,.$$

Note that $\rho_0 = \varnothing$ and $|\rho_r| = r$ for all $r \geqslant 1$, since $\mathbb{D}(q_0) = \varnothing$ and each variable occurs only once in s_0. The next claim can be seen as the core of the proof, as for the proof of Theorem 25(b).

Claim 29. *For every $r \in \{0, 1, \ldots, \hat{r}\}$ we have $q_r \sim_{\rho_r} q'$.*

Before turning to the proof of Claim 29 let us first argue how the claim serves for completing the proof of Theorem 25(b). For $r = \hat{r}$ we know that $V^{\mathrm{u}}_{\hat{r}} = \varnothing$. Hence, $\{x_1, \ldots, x_{\hat{r}}\} = \mathbb{V}(q_0)$ and $\rho_{\hat{r}} = [k \cdot \ell]$. From Claim 29 we obtain $q_{\hat{r}} \sim_{[k \cdot \ell]} q'$. But, by Lemma 19 this implies that $q_{\hat{r}} \cong q'$, contradicting our assumption that $q_{\hat{r}} \not\equiv q'$. Thus, all that remains to complete the proof of Theorem 25(b) is to prove Claim 28 (at the end of the proof of the theorem) and Claim 29.

Proof of Claim 29. We proceed by induction on r. For the induction base with $r = 0$ recall that $q_0 = q_{mg}$. Thus, $\rho_0 = \varnothing$, which immediately implies that $q_0 \sim_{\rho_0} q'$.

For the induction step consider an arbitrary $r \geqslant 1$. At the beginning of the r-th iteration of the main loop of the main algorithm the situation is as follows: $V^{\mathrm{u}} = V^{\mathrm{u}}_{r-1} \neq \varnothing$ and $s = s_{r-1}$ and, by Claim 28(1), $V^{\mathrm{u}}_{r-1} = \mathbb{V}(q_0) \setminus \{x_\nu \;:\; 1 \leqslant \nu \leqslant r-1\}$. Recall that by x_r we denote the particular element of V^{u}_{r-1} chosen at the beginning of the r-th iterations through the main loop. We have to prove the following:

1. For all $(i,u), (j,v) \in \rho_r$ we have

$$s_r[i][u] = s_r[j][v] \Leftrightarrow s'[i][u] = s'[j][v],$$

2. for all $(i,u) \in \rho_r$ we have

$$s_r[i][u] \in \mathbb{I} \cup \mathbb{D} \Rightarrow s'[i][u] = s_r[i][u] \quad \text{and} \quad s'[i][u] \in \mathbb{I} \cup \mathbb{D} \Rightarrow s_r[i][u] = s'[i][u]$$

The induction hypothesis states that $q_{r-1} \sim_{\rho_{r-1}} q'$ holds. We have to show that $q_r \sim_{\rho_r} q'$ holds as well, where $\rho_r = \rho_{r-1} \cup pos(s_0, x_r)$. From (†) we know that $q_r \xrightarrow{\text{hom}} q'$. Thus, the following is true:

1. For all $(i,u),(j,v) \in [\ell] \times [k]$: $s_r[j][v] = s_r[i][u] \in \mathbb{V}$ then $s'[j][v] = s'[i][u]$.
2. For all $(i,u) \in [\ell] \times [k]$: If $s_r[i][u] \in \mathbb{I} \cup \mathbb{D}$ then $\varnothing \subset s'[i] \subseteq s_r[i]$.

Furthermore, by Claim 28(5), s_r coincides with s_{r-1} on all positions $(i,u) \in [\ell] \times [k]$ with $(i,u) \notin pos(s_0, x_r)$.

Recall that $|pos(s_0, x_r)|$=1 due to $s_0 = s_{mg}$. To ease notation we simply write (r_i, r_u) to denote the unique position of x_r in s_0. Since $(r_i, r_u) \notin \rho_{r-1}$, the induction hypothesis $q_{r-1} \sim_{\rho_{r-1}} q'$ hence implies that $q_r \sim_{\rho_{r-1}} q'$, and therefore also $s_r[j][v] = s_r[i][u] \Leftrightarrow s'[j][v] = s'[i][u]$ for all $(i,u),(j,v), \in \rho_{r-1}$. In order to prove that $q_r \sim_{\rho_r} q'$, it only remains to prove the following:

(i) For all $(j,v) \in \rho_{r-1}$: $s_r[j][v], s_r[r_i][r_u] \in \mathbb{I} \cup \mathbb{D}$ and $s_r[j][v] = s_r[r_i][r_u]$ then $s'[j][v] = s'[r_i][r_u]$.

(ii) For all $(j,v) \in \rho_{r-1}$: If $s'[j][v] = s'[r_i][r_u]$ then $s_r[j][v] = s_r[r_i][r_u]$.

(iii) If $s'[r_i][r_u] \in \mathbb{I} \cup \mathbb{D}$ then $s_r[r_i][r_u] \subseteq s'[r_i][r_u]$.

Let n denote the attribute index of the variable x_r that is considered for replacement. By the definition of the main algorithm, the query q_r is obtained from q_{r-1} by performing exactly

- one replacement operation using an interval (InterOp) in Line 14 of Function 3 with $I \subset [\min(adom_n(\mathcal{S})), \max(adom_n(\mathcal{S}))]$ if $n \in O$,
- one replacement operation using a disjunction (DisjOp) in Line 5 of Function 2 with $y \in \Omega \setminus V^{\mathrm{a}} = \Delta^i_{A_n} \setminus V^{\mathrm{a}} \in \Delta$ for $i \in [d]$ if $n \in U$,
- one replacement operation using a variable (VarOp) in Line 5 of Function 2 (called by Line 17 or 13 of the main algorithm) with $y \in V^{\mathrm{a}}$,
- or no replacement operation (NoChangeOp) in Line 22 of the main algorithm.

Note that a NoChangeOp will only be performed if the following conditions are true:

$$\text{If } n \in O, \text{ then for every interval } I \in \mathbb{D} \text{ as decribed above the query } q_I \coloneqq q_{r-1}\langle x_r \mapsto I\rangle \text{ does not satisfy } \mathsf{supp}(q_I, \mathcal{S}) \geq \mathsf{sp}. \qquad (*)_r$$

and

$$\text{If } n \in U, \text{ for every disjunction } \chi \in \bigcup_{i \in [\min(d, |adom_n(\mathcal{S})|)-1]} \Delta^i_{A_n} \text{ the query } q_\chi \coloneqq q_{r-1}\langle x_r \mapsto \chi\rangle \text{ does not satisfy } \mathsf{supp}(q_\chi, \mathcal{S}) \geq \mathsf{sp}. \qquad (**)_r$$

and

For every variable $y \in V^{\mathrm{a}}_{r-1}$ the query $q_y \coloneqq q_{r-1}\langle x_r \mapsto y\rangle$ does not satisfy $\mathsf{supp}(q_y, \mathcal{S}) \geq \mathsf{sp}$. $(***)_r$

Claim 30. *Let (r_i, r_u) be the position of x_r in s_0.*

- *Depending on whether $r_u \in O$, $(*)_r$ or $(**)_r$ (in case that $r_u \notin O$) imply that $s'[r_i][r_u] \in \mathbb{V}$.*
- *$(***)_r$ implies that $s'[r_i][r_u] \in \mathbb{I} \cup \mathbb{D}$ or $s'[r_i][r_u] \neq s'[j][v]$ for all $(j, v) \in \rho_{r-1}$.*

Proof. Let us first focus on the claim's first statement for the case that $r_u \in O$. Let (r_i, r_u) be the position of x_r in s_0. Let $(*)_r$ be satisifed if $r_u \in O$. This implies that there is no $I \subset [\min(adom_{r_u}(\mathcal{S})), \max(adom_{r_u}(\mathcal{S}))]$ such that replacing x_r by I yields a query that satisfies sp. For contradiction, assume $s'[r_i][r_u] = I \in \mathbb{I}$. By the definition of Function 3, and since $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$, we know that $I \subset [\min(adom_{r_u}(\mathcal{S})), \max(adom_{r_u}(\mathcal{S}))]$. For $q_I \coloneqq q_{r-1}\langle x_r \mapsto I\rangle$ we have $q_I \xrightarrow{\text{hom}} q'$, because $q_{r-1} \xrightarrow{\text{hom}} q'$ and $s'[r_i][r_u] = I$. Therefore, $\mathsf{supp}(q_I, \mathcal{S}) \geqslant \mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$, contradicting $(**)_r$. The proof for the case that $r_u \notin O$ can be handeled analogously. This completes the proof of the first statement.

Let us now turn to the second statement of Claim 30. Let (r_i, r_u) be the position of x_r in s_0 and let $(***)_r$ be satisfied, i. e. there exists no available variable $y \in V^{\mathrm{a}}_{r-1}$ such that replacing x_r by y yields a query that satisfies sp. If $s'[r_i][r_u] \in I \cup \mathbb{D}$ we are done. Consider the case where $s'[r_i][r_u] \in \mathbb{V}$ and assume for contradiction that there exists a position $(m, n) \in \rho_{r-1}$ such that $s'[r_i][r_u] = s'[m][n]$. Let $y \coloneqq s_{r-1}[m][n]$. From $q_{r-1} \xrightarrow{\text{hom}} q'$ and $s'[m][n] \in \mathbb{V}$ we obtain $y \in \mathbb{V}$. Since $(m, n) \in \rho_{r-1}$ we then obtain that there is a $\nu \in \{1, \ldots, r-1\}$ such that $(m, n) \in pos(s_0, x_\nu)$. We claim that $x_\nu \in V^{\mathrm{a}}_{r-1}$. For contradiction, assume that $x_\nu \notin V^{\mathrm{a}}_{r-1}$. By Claim 28(1) we have $\mathbb{V}(s_{r-1}) = V^{\mathrm{u}}_{r-1} \cup V^{\mathrm{a}}_{r-1}$, and hence $y \in V^{\mathrm{u}}_{r-1}$. From Claim 28(4) we obtain that $pos(s_{r-1}, y) = pos(s_0, y)$. Hence, $\{(m, n)\} = pos(s_0, y) \cap pos(s_0, x_\nu)$. This implies that $y = x_\nu$ and due to $x_\nu \in \{x_1, \ldots, x_{r-1}\}$ it holds that $y \in \{x_1, \ldots, x_{r-1}\}$. But this is a contradiction to $y \in V^{\mathrm{u}}_{r-1} = \mathbb{V}(q_0) \setminus \{x_1, \ldots, x_{r-1}\}$. Thus, we have shown that $y \in V^{\mathrm{a}}_{r-1}$.

Consider the query $q_y \coloneqq q_{r-1}\langle x_r \mapsto y\rangle$, and let s_y be the query string of q_y. It holds that $q_y \xrightarrow{\text{hom}} q'$ since we already know that $q_{r-1} \xrightarrow{\text{hom}} q'$. i. e., there is a homomorphism $h\colon ((\![\ell]\!] \times [\![k]\!]) \times (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})) \to (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ from q_{r-1} to q'. This h also is a homomorphism from q_y to q'. To see this, note that for $(r_i, r_u) \in pos(s_{r-1}, x_r)$ we have $h(s_y[r_i][r_u]) = h(y) = h(s_{r-1}[m][n]) = s'[m][n] = s'[r_i][r_u]$; and for every other position $(j, v) \in [\![\ell]\!] \times [\![k]\!] \setminus pos(s_{r-1}, x_r)$ we have $h(s_y[j][v]) = h(s_{r-1}[j][v]) = s'[j][v]$. Therefore, $\mathsf{supp}(q_y, \mathcal{S}) \geqslant \mathsf{sp}$ which contradicts $(***)_r$. This ends the proof of Claim 30. □

To complete the proof of Claim 29 we now distinguish between the four cases depending on whether the query q_r is obtained from q_{r-1} by performing a InterOp, DisjOp, VarOp, or a

NoChangeOp. Our aim to show that in all cases the conditions (i) to (iii) are satisfied. Let us briefly recall these conditions:

(i) For all $(j, v) \in \rho_{r-1}$: $s_r[j][v], s_r[r_i][r_u] \in \mathbb{I} \cup \mathbb{D}$ and $s_r[j][v] = s_r[r_i][r_u]$ then $s'[j][v] = s'[r_i][r_u]$.

(ii) For all $(j, v) \in \rho_{r-1}$: If $s'[j][v] = s'[r_i][r_u]$ then $s_r[j][v] = s_r[r_i][r_u]$.

(iii) If $s'[r_i][r_u] \in \mathbb{I} \cup \mathbb{D}$ then $s_r[r_i][r_u] \subseteq s'[r_i][r_u]$.

Case 1: q_r is obtained from q_{r-1} by replacing the current variable x_r in s_{r-1} by an interval $I \in \mathbb{I}$, i. e. $s_r = s_{r-1}\langle x_r \mapsto I\rangle$. Let (r_i, r_u) the position of x_r in s_{r-1}. By (2) we have $s'[r_i][r_u] \subseteq s[r_i][r_u] = I$. This interval I is minimal in the sense that for each $I' \in \mathbb{I}$ with $I' \subset I$ it holds that $\mathsf{supp}(q_{r-1}\langle x_r \mapsto I'\rangle, \mathcal{S}) < \mathsf{sp}$, due to Function 3: the initial interval $[\min(adom_{r_u}(\mathcal{S})), \max(adom_{r_u}(\mathcal{S}))]$ decreases during each iteration of the while-loop in Line 4, until a smaller interval does not satisfy the support threshold any more, or the current interval is already a singleton. Hence, $s_r[r_i][r_u] = I \subseteq s'[r_i][r_u]$ holds as well, since otherwise $s'[r_i][r_u] = I' \subset I$ which contradicts $\mathsf{supp}(q_r, \mathcal{S}) \geqslant \mathsf{sp}$. In particular (iii) is satisfied.

To see that (ii) is satisfied, consider $(j, v) \in \rho_{r-1}$ with $s'[r_i][r_u] = s'[j][v] = I$. By (2) we have $s_r[r_i][r_u] \subseteq s'[r_i][r_u]$ and $s_r[j][v] \subseteq s'[j][v]$. Due to $q_{r-1} \xrightarrow{\text{hom}} q'$ it holds that $s'[j][v] \subseteq s_r[j][v]$, and thanks to $q_r \xrightarrow{\text{hom}} q'$ we have $s'[r_i][r_u] \subseteq s_r[r_i][r_u]$ since $s_r[r_i][r_u] \in \mathbb{I}$. Hence, (ii) is satisfied.

It remains to prove that (i) is satisfied. For contradiction assume that $s_r[j][v] = s_r[r_i][r_u] = I \in \mathbb{I}$, and $s'[j][v] = I_1 \neq I_2 = s'[r_i][r_u]$. By $q_r \xrightarrow{\text{hom}} q'$ it holds that $I_1, I_2 \in \mathbb{I}$, and $I_1, I_2 \subseteq I$. Assume that $I_1 \subset I$. Note that all other cases can be dealt with analogously, except the case that $I_1 = I_2 = I$ which contradicts our assumption. Assume that I was found to match the support threshold, i. e. $\mathsf{supp}(q_{r-1}\langle x_r \mapsto I\rangle, \mathcal{S}) \geqslant \mathsf{sp}$ during the i-th iteration of the while-loop of Function 3 by satisfying the support threshold in Line 6, 8, or 10. Furthermore, assume that I was selected for the replacement operation after the j-th iteration, $j \geqslant i$ of the while-loop. Note that is may take further iterations through the while-loop in order to achieve the termination criterion $\mathtt{lowb} - n \leqslant 1$ and $u - ub \leqslant 1$. During these iterations, subintervals I' of I have been checked and were discarded since $\mathsf{supp}((s_{r-1}\langle x_r \mapsto I'\rangle, w, c), \mathcal{S}) < \mathsf{sp}$. At least one of these subintervals may either equal I_1 or strictly contain I_1. Hence, $\mathsf{supp}((s_{r-1}\langle x_r \mapsto I_1\rangle, w, c), \mathcal{S}) < \mathsf{sp}$. Note that q' is a specialisation of q_{r-1} due to $q_{r-1} \sim_{\rho_{r-1}} q'$, and $q_{r-1} \xrightarrow{\text{hom}} q'$. In case that $\mathsf{supp}((s_{r-1}\langle x_r \mapsto I_1\rangle, w, c), \mathcal{S}) < \mathsf{sp}$, we already know that $\mathsf{supp}((\tilde{q}, w, c), \mathcal{S}) < \mathsf{sp}$ for each query $\tilde{q}$ that is a specialisation of $(s_{r-1}\langle x_r \mapsto \chi_1\rangle, w, c)$, including q'. Hence, $\mathsf{supp}(q', \mathcal{S}) < \mathsf{sp}$, contradicting the assumption.

Case 2: q_r is obtained from q_{r-1} by replacing the current variable x_r in s_{r-1} by a disjunction $\chi \in \Delta$, i. e. $s_r = s_{r-1}\langle x_r \mapsto \chi\rangle$. Let (r_i, r_u) the position of x_r in s_{r-1}. By

(2) we have $\varnothing \subset s'[r_i][r_u] \subseteq s[r_i][r_u] = \chi$. Since Δ is walked through incrementally in Line 11 χ is minimal in the following sense: for each $\chi' \in \Delta$ with $\chi' \subset \chi$ it holds that $\mathsf{supp}(q_{r-1}\langle x_r \mapsto \chi'\rangle) < \mathsf{sp}$. Hence, $s[r_i][r_u] = \chi \subseteq s'[r_i][r_u]$ holds as well, since otherwise $s'[r_i][r_u] = \chi' \subset \chi$ which contradicts $\mathsf{supp}(q_r, \mathcal{S}) \geqslant \mathsf{sp}$. In particular (iii) is satisfied.

To see that (ii) is satisfied, consider $(j, v) \in \rho_{r-1}$ with $s'[r_i][r_u] = s'[j][v] = \chi$. By (2) we have $\varnothing \subset s_r[r_i][r_u] \subseteq s'[r_i][r_u]$ and $\varnothing \subset s_r[j][v] \subseteq s'[j][v]$. Due to $q_{r-1} \xrightarrow{\text{hom}} q'$ it holds that $s'[j][v] \subseteq s_r[j][v]$, and thanks to $q_r \xrightarrow{\text{hom}} q'$ we have $s'[r_i][r_u] \subseteq s_r[r_i][r_u]$ since $s_r[r_i][r_u] \in \Delta$. Hence, (ii) is satisfied.

It remains to prove that (i) is satisfied. For contradiction assume that $s_r[j][v] = s_r[r_i][r_u] = \chi \in \Delta$, and $s'[j][v] = \chi_1 \neq \chi_2 = s'[r_i][r_u]$. By $q_r \xrightarrow{\text{hom}} q'$ it holds that $\chi_1, \chi_2 \in \mathbb{D}$, and $\chi_1, \chi_2 \subseteq \chi$. Assume that $\chi_1 \subset \chi$. Note that all other cases can be dealt with analogously, except the case that $\chi_1 = \chi_2 = \chi$ which contradicts our assumption. Let $i = |\chi|$. Hence, χ was selected for the replacement operation during the i-th iteration of the for-loop, i. e. $\chi \in \Delta^i$. Since $\chi_1 \subset \chi$ it either holds that $\chi_1 \in \Delta^{i'}$ for $i' < i$, or $\chi_1 \notin \Delta$. In case of the latter, χ_1 is not a supported disjunction, which implies that q' does not cover the given sample with the requested support, contradicting $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$. Otherwise, Algorithm 1 considers χ_1 during the i'-th iteration of the for-loop and discards it. This happens if $\mathsf{supp}((s_{r-1}\langle x_r \mapsto \chi_1\rangle, w, c), \mathcal{S}) < \mathsf{sp}$. Note that q' is a specialisation of q_{r-1} due to $q_{r-1} \sim_{\rho_{r-1}} q'$, and $q_{r-1} \xrightarrow{\text{hom}} q'$. In case that $\mathsf{supp}((s_{r-1}\langle x_r \mapsto \chi_1\rangle, w, c), \mathcal{S}) < \mathsf{sp}$, we already know that $\mathsf{supp}((\tilde{q}, w, c), \mathcal{S}) < \mathsf{sp}$ for each query $\tilde{q}$ that is a specialisation of $(s_{r-1}\langle x_r \mapsto \chi_1\rangle, w, c)$, including q'. Hence, $\mathsf{supp}(q', \mathcal{S}) < \mathsf{sp}$, contradicting the assumption.

Case 3: q_r is obtained from q_{r-1} by replacing the variable x_r in s_{r-1} by an available variable $y \in V^{\text{a}}_{r-1}$, i. e. the query string of q_r is $s_r = s_{r-1}\langle x_r \mapsto y\rangle$. According to Claim 28(1) there exists an $r' \leqslant r - 1$ such that $y = x_{r'}$. Furthermore, by definition of the algorithm, a variable can only be included into the set V^{a} in case of a NoChangeOp, i. e. neither an InterOp, a DisjOp nor a VarOp was possible. Therefore, in the r'-th iteration of the mains algorithm's while-loop, the variable $x_{r'}$ was included into the set V^{a}. But this means that the conditions $(*)_r$, $(**)_r$ and $(***)_r$ are satisfied. Let $\{(r'_i, r'_u)\} \in pos(s_0, x_{r'})$. The first statement of Claim 30 tells us that $s'[r'_i][r'_u] \in \mathbb{V}$. Note that $(r'_i, r'_u) \in \rho_{r'} \subseteq \rho_{r''}$ for all $r'' \geqslant r'$. Hence, by Claim 28(5) we obtain that $y = x_{r'} = s_{r'}[r'_i][r'_u] = s_{r''}[r'_i][r'_u]$ for all $r'' \geqslant r'$. In particular, for $r'' = r$ we obtain that $y = s_r[r'_i][r'_u]$. Hence, we have $s_r[r'_i][r'_u] = y = s_r[r_i][r_u]$ for $(r_i, r_u) \in pos(s_0, x_r)$. From $q_r \xrightarrow{\text{hom}} q'$ we obtain that $s'[r'_i][r'_u] = s'[r_i][r_u]$. Since $s'[r'_i][r'_u] \in \mathbb{V}$ we obtain that $s'[r_i][r_u] \in \mathbb{V}$.
Now we can turn to condition (ii). Let $(r_i, r_u) \in pos(s_0, x_r)$ and choose an arbitrary $(j, v) \in \rho_{r-1}$ such that $s'[j][v] = s'[r_i][r_u]$. We want to prove that $s_r[j][v] = s_r[r_i][r_u]$. As shown above, $s'[j][v] = s'[r_i][r_u] = s'[r'_i][r'_u]$. From $(j, v), (r'_i, r'_u) \in \rho_{r-1}$ and

$q_r \sim_{\rho_{r-1}} q'$ we obtain that $s_r[j][v] = s_r[r'_i][r'_u]$. And we already know that $s_r[r'_i][r'_u] = y = s_r[r_i][r_u]$. This proves condition (ii).

Note that conditions (i) and (iii) are trivially satisifed.

Case 4: q_r is obtained from q_{r-1} by performing no replacement operation at all. In this case we know that the statements $(**)_r$ and $(***)_r$ are satisfied. From Claim 30 we obtain for $(r_i, r_u) \in pos(s_0, x_r)$ that $s'[r_i][r_u] \in \mathbb{V}$ and $s'[j][v] \neq s'[r_i][r_u]$ for all $(j, v) \in \rho_{r-1}$. Hence, (i) to (iii) are trivially satisifed.

In all three cases we have shown that (i) to (iii) are satisfied, and thus we have $q_r \sim_{\rho_r} q'$. This completes the proof of Claim 29. ■

In summary the proof of of (b) is now completed. □

Claim 31 (restated). *For every $r \geqslant 1$ we have*

1. $V_r^{\mathrm{u}} = \mathbb{V}(q_0) \setminus \{x_1, \ldots, x_r\}$ *and* $V_r^{\mathrm{u}} \cap V_r^{\mathrm{a}} = \varnothing$ *and* $V_r^{\mathrm{u}} \cup V_r^{\mathrm{a}} = \mathbb{V}(s_r)$.
2. $\mathbb{D}(q_r) \subseteq \Delta$ *and* $\mathsf{supp}(q_r, \mathcal{S}) \geqslant \mathsf{sp}$.
3. $q_{r-1} \xrightarrow{\text{hom}} q_r$.
4. *For every* $x \in V_r^{\mathrm{u}}$ *we have* $pos(s_r, x) = pos(s_0, x)$.
5. $s_r[i][v] = s_{r-1}[i][v]$ *for all* $(i, v) \in \big([\ell] \times [k]\big) \setminus pos(s_0, x_r)$.

Proof of Claim 28 by induction on r. Base case: (r = 1) Let x_1 be the chosen variable during the first iteration through the while-loop of Algorithm 1. At the end of the first iteration through the while-loop either an InterOp, a DisjOp, or a NoChangeOp has been executed. Note that a VarOp is not possible due to $V_0 = \varnothing$.

1. In all three cases it holds that $V_1^{\mathrm{u}} = V_0^{\mathrm{u}} \setminus \{x_1\} = \mathbb{V}(q_0) \setminus \{x_1\}$, since the chosen variable is deleted from the set of unvisited variables in Line 7.
   Furthermore, $V_1^{\mathrm{u}} \cap V_1^{\mathrm{a}} = \varnothing$, because either an InterOp or a DisjOp has been executed, implying that $V_1 = \varnothing$, or $V_1 = \{x_1\}$ and $x_1 \notin V_1^{\mathrm{u}}$ (due to Line 7) if a NoChangeOp was executed.
   Because $U_0 = \mathbb{V}(q_0)$ and $V_0 = \varnothing$ it holds that $V_0^{\mathrm{u}} \cup V_0^{\mathrm{a}} = \mathbb{V}(s_0)$. If a NoChangeOp has been executed, we conclude that $V_1^{\mathrm{u}} \cup V_1^{\mathrm{a}} = (V_0^{\mathrm{u}} \setminus \{x_1\}) \cup V_0^{\mathrm{a}} \cup \{x_1\} = \mathbb{V}(q_0) = \mathbb{V}(q_1)$. Otherwise, in case of an InterOp or a DisjOp, $V_1^{\mathrm{u}} \cup V_1^{\mathrm{a}} = (V_0^{\mathrm{u}} \setminus \{x_1\}) \cup V_0^{\mathrm{a}} = V_0^{\mathrm{u}} \setminus \{x_1\} = \mathbb{V}(q_0) \setminus \{x_1\} = \mathbb{V}(q_1)$.
2. In case of a NoChangeOp it holds that $\mathbb{D}(q_1) = \mathbb{D}(q_0) = \varnothing \subseteq \Delta$ since $q_0 = q_{mg}$ and $\mathbb{D}(q_{mg}) = \varnothing$. If instead the (single) occurrence of x_1 has been replaced by a disjunction χ, then, due to Line 13, this disjunction is chosen from $\Omega = \Delta_i \subseteq \Delta$, for $i \in [d]$. Therefore,

$$\mathbb{D}(q_1) = \mathbb{D}(q_0) \cup \chi \overset{\chi \in \Delta_i,\ \mathbb{D}(q_0) \subseteq \Delta}{\subseteq} \Delta.$$

If $q_1 = q_0$ (after a NoChangeOp), it holds that $\mathsf{supp}(q_1, \mathcal{S}) = \mathsf{supp}(q_0, \mathcal{S}) \geqslant \mathsf{sp}$. Otherwise, q_1 is obtained from q_0 by an InterOp or a DisjOp, which replaced x_1 by some $I \in \mathbb{I}$ or $\chi \in \Omega = \Delta_i$, respectively. Note that this operation is only applied if the query $q_1 = (s_0(x_1 \mapsto I), w, c)$ or $q_1 = (s_0(x_1 \mapsto \chi), w, c)$, respectively, satisfies the given support-threshold, i. e. $\mathsf{supp}(q_1, \mathcal{S}) \geqslant \mathsf{sp}$.

3. After a NoChangeOp $q_1 = q_0$, hence the identity is a homomorphism from q_0 to q_1. Otherwise, q_1 was obtained from q_0 by replacing x_1 by an interval $I \subset \mathbb{I}$ or a disjunction $\chi \in \Omega = \Delta_i \subseteq \Delta$, for $i \in [d]$. The mapping $h : (([\ell] \times [k]) \times \mathbb{V} \cup \mathbb{I} \cup \mathbb{D}) \to (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ such that $h(x_1) = I$ or $h(x_1) = \chi$, and h is the identity on $\mathbb{V} \cup \mathbb{I} \cup \mathbb{D} \setminus \{I\}$, or $\mathbb{V} \cup \mathbb{I} \cup \mathbb{D} \setminus \{\chi\}$, respectively, is a homomorphism from q_0 to q_1. (Note that x_1 only occurs once in s_0.)

4. In case of a NoChangeOp, we have $pos(s_1, x) = pos(s_0, x)$ for every $x \in V_1^{\mathrm{u}}$, since the query string has not changed, i.e. $q_1 = q_0$. It remains to show that this holds after an InterOp or a DisjOp, too. For contradiction, assume that there exists a $x \in V_1^{\mathrm{u}}$ such that $pos(s_1, x) \neq pos(s_0, x)$. Since each InterOp or DisjOp only changes the positions carrying the currently selected variable (for the base case this is x_1), while all other positions remain unchanged (due to Line 14 in Function 3 and Line 5 in Function 2), this implies that $x = x_1$ and therefore, $x \notin V_1^{\mathrm{u}}$, contradicting the assumption.

5. After a NoChangeOp it holds that $s_1[j][v] = s_0[j][v]$ for all $(j, v) \in ([\ell] \times [k])$ and especially for all $(j, v) \in ([\ell] \times [k]) \setminus pos(s_0, x_1)$. In case that an InterOp or a DisjOp was performed, assume for contradiction that there exists a position $(j, v) \in ([\ell] \times [k]) \setminus pos(s_0, x_1)$ such that $s_1[j][v] \neq s_0[j][v]$ after the replacement operation. By construction of the algorithm only replaces the occurrences of x_1 by some interval I or disjunction χ in Line 14 of Function 3 or Line 5 in Function 2, during an InterOp or a DisjOp, respectively. Hence, (j, v) has to be in $pos(s_0, x_1)$, which contradicts the assumption.

Inductive step: $(r - 1 \rightsquigarrow r)$ Let x_r be the chosen variable during the r-th iteration through the outer while-loop. At the end of this iteration either InterOp, DisjOp, VarOp or NoChangeOp has been executed.

1. Assume that the induction hypothesis holds for $r - 1$, i.e.

$$V_{r-1}^{\mathrm{u}} = \mathbb{V}(q_0) \setminus \{x_1, \ldots, x_{r-1}\} \tag{1a}$$
$$V_{r-1}^{\mathrm{u}} \cap V_{r-1}^{\mathrm{a}} = \varnothing \tag{1b}$$
$$V_{r-1}^{\mathrm{u}} \cup V_{r-1}^{\mathrm{a}} = \mathbb{V}(s_{r-1}) \tag{1c}$$

No matter which kind of operation has been executed during the r-th iteration through the outer while-loop, $V_r^{\mathrm{u}} = V_{r-1}^{\mathrm{u}} \setminus \{x_r\}$ holds, due to the deletion of the selected unvisited variable from the set of unvisited variables in Line 7. Using the induction hypothesis 1a we can conclude that

$$\begin{aligned} V_r^{\mathrm{u}} &= V_{r-1}^{\mathrm{u}} \setminus \{x_r\} \\ &\overset{\text{i.h.}}{=} \mathbb{V}(q_0) \setminus \big(\{x_1, \ldots, x_{r-1}\} \cup \{x_r\}\big) \\ &= \mathbb{V}(q_0) \setminus \{x_1, \ldots, x_r\}. \end{aligned}$$

After the r-th iteration through the while-loop, V_r^{a} equals either $V_{r-1}^{\mathrm{a}} \cup \{x_r\}$ (if a NoChangeOp has been executed) or V_{r-1}^{a}. In the latter case it holds that

$$V_r^{\mathrm{u}} \cap V_r^{\mathrm{a}} = V_{r-1}^{\mathrm{u}} \setminus \{x_r\} \cap V_{r-1}^{\mathrm{a}} \overset{\text{i.h.}}{=} \varnothing$$

and

$$V_r^{\mathrm{u}} \cup V_r^{\mathrm{a}} = V_{r-1}^{\mathrm{u}} \setminus \{x_r\} \cup V_{r-1}^{\mathrm{a}} \overset{\text{i.h.}}{=} \mathbb{V}(s_{r-1}) \setminus \{x_r\} = \mathbb{V}(s_r).$$

Otherwise, in case a NoChangeOp has been executed, it holds that

$$V_r^{\mathrm{u}} \cap V_r^{\mathrm{a}} = \left(V_{r-1}^{\mathrm{u}} \setminus \{x_r\}\right) \cap \left(V_{r-1}^{\mathrm{a}} \cup \{x_r\}\right) \overset{\text{i.h.}}{=} \varnothing$$

and

$$V_r^{\mathrm{u}} \cup V_r^{\mathrm{a}} = V_{r-1}^{\mathrm{u}} \setminus \{x_r\} \cup V_{r_1}^{\mathrm{a}} \cup \{x_1\} \overset{\text{i.h.}}{=} \mathbb{V}(s_{r-1}) = \mathbb{V}(s_r).$$

2. Assume that the induction hypothesis holds for $r-1$, i. e. $\mathbb{D}(q_{r-1}) \subseteq \Delta$ and $\mathsf{supp}(q_{r-1}, \mathcal{S}) \geqslant \mathsf{sp}$. If a NoChangeOp has been executed, $q_r = q_{r-1}$ and we are done thanks to the induction hypothesis.
Consider the case that a DisjOp has been executed during the r-th iteration through the outer while-loop. If x_r was replaced by a disjunction $\chi \in \mathbb{D}(q_{r-1})$ we conclude that $\mathbb{D}(q_r) = \mathbb{D}(q_{r-1}) \subseteq \Delta$ by the induction hypothesis. Otherwise, it holds that $\mathbb{D}(q_r) = \mathbb{D}(q_{r-1}) \cup \chi \subseteq \Delta$, as well by the induction hypothesis and the fact that $\chi \in \Omega = \Delta_i \subseteq \Delta$, for $i \in [d]$, by construction of the algorithm. Note that $\mathbb{D}(q_r) = \mathbb{D}(q_{r-1}) \subseteq \Delta$ holds by the induction hypothesis in case of a VarOp.
In the case that q_r is the result of an InterOp, a DisjOp or a VarOp it holds that q_r satisfies the support, due to the construction of the algorithm: an InterOp, a TypeOp or VarOp is only applied if the corresponding requests to the oracle in evaluates to true (Line 6, 8, or 10 of Function 3, or Line 4 of Function 2).

3. After a NoChangeOp $q_r = q_{r-1}$, hence the identity is a homomorphism from q_{r-1} to q_r. Otherwise, q_r was obtained from q_{r-1} by replacing x_r by an interval $I \in \mathbb{I}$, a disjunction $\chi \in \Omega = \Delta_i$, for $i \in [d]$, or a variable $x \in V_{r-1}^{\mathrm{a}}$. The mapping $h : (([\ell] \times [k]) \times \mathbb{V} \cup \mathbb{I} \cup \mathbb{D}) \to (\mathbb{V} \cup \mathbb{I} \cup \mathbb{D})$ such that $h(x_r) = I$, $h(x_r) = \chi$ or $h(x_r) = x$ and h is the identity on $\mathbb{V} \cup \mathbb{I} \cup \mathbb{D} \setminus \{I\}$, $\mathbb{V} \cup \mathbb{I} \cup \mathbb{D} \setminus \{\chi\}$ or $\mathbb{V} \cup \mathbb{I} \cup \mathbb{D} \setminus \{x_r\}$, in case of an InterOp, a DisjOp or VarOp, respectively, is a homomorphism from q_{r-1} to q_r. Note that $h(x_r) = I$ or $h(x_r) = \chi$ does not contradict our definition of homomorphisms, since x_r does only occur once within s_{r-1}.

4. Assume that the induction hypothesis holds for $r-1$, i.e. for every $x \in V_{r-1}^{\mathrm{u}}$ we have $pos(s_{r-1}, x) = pos(s_0, x)$. If a NoChangeOp has been executed, the query string remains the same. By the induction hypothesis $pos(s_r, x) = pos(s_{r-1}, x) = pos(s_0, x)$ for every $x \in V_{r-1}^{\mathrm{u}}$ and hence, $pos(s_r, x) = pos(s_0, x)$ for every $x \in V_r^{\mathrm{u}}$.
Otherwise, an InterOp, a DisjOp or a VarOp has changed the single position in $pos(s_{r-1}, x_r)$. As already discussed the algorithm cannot change any other position in the query string. Furthermore, $x_r \notin V_r^{\mathrm{u}}$. Hence, for every $x \in V_r^{\mathrm{u}}$ (note that $x \neq x_r$) holds $pos(s_r, x) = pos(s_{r-1}, x) \overset{\text{i.h.}}{=} pos(s_0, x)$.

5. Assume that the induction hypothesis holds for $r-1$, i.e. $s_{r-1}[j][v] = s_{r-2}[j][v]$ for all $(j,v) \in ([\ell] \times [k]) \setminus pos(s_0, x_{r-1})$. If a NoChangeOp has been executed, the query string remains the same, which implies $s_r[j][v] = s_{r-1}[j][v]$ for all $(j,v) \in ([\ell] \times [k])$ and especially for all $(j,v) \in ([\ell] \times [k]) \setminus pos(s_0, x_r)$. Otherwise, assume for contradiction, that there exists a position $(j,v) \in ([\ell] \times [k]) \setminus pos(s_0, x_r)$ such that $s_r[j][v] \neq s_{r-1}[j][v]$ after an InterOp, a DisjOp or a VarOp. Due to the induction hypothesis, this inequality appeared just during the r-th iteration of the while-loop of the main algorithm. By construction of the algorithm and its functions, actually Line 14 of Function 3 or Line 5 in Function 2, only the position $(j',v') \in pos(s_0, x_r)$ is replaced. This implies $(j,v) = (j',v')$, and therefore, $(j,v) \in pos(s_0, x_r)$ contradicting the assumption.

□