

Puzzling over Subsequence-Query Extensions: Disjunction and Generalised Gaps

André Frochaux^{1,†}, Sarah Kleest-Meißner^{1,†}

¹Humboldt-Universität zu Berlin, Unter den Linden 6, 10117 Berlin, Germany

Abstract

A query model for sequence data was introduced in [1] 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 types, as well as a global window size and a number of local gap-size constraints. We propose two new extensions of swg-queries, which both enrich the expressive power of swg-queries in different ways: subsequence-queries with generalised gap-size constraints (swgg-queries, for short) and disjunctive subsequence-queries (dswg-queries, for short). We discuss a suitable characterisation of containment, a classical property considered in database theory, and adapt results concerning the discovery of swg-queries to both, swgg-queries and dswg-queries.

Keywords

subsequence-queries, disjunction, gap-constraints, learning descriptive queries, subsequences, embeddings

1. Introduction

Applications in different domains like cluster monitoring [2], urban transportation [3], and in finance[4], use models for sequence data, which define an order for a set of data items [5]. Respective systems enable the definition of queries which detect patterns of data items describing a *situation of interest* (*soi* for short), for example error occurrence, in a specific order and temporal context.

Finding a suitable query is a non-trivial task. A user may know the time at which a certain job fails execution, but does not exactly conceive a situation which forecasts the failure. It was therefore suggested to automatically discover a query from historic sequence data which describes the *soi*. Such a query may then be used in pro-active applications where they shall anticipate a *soi* to prepare for it accordingly [6].

In [1] a formal model (referred to as *swg-queries*) was proposed, which covers the essence of discovering a query from sequence data. In a nutshell, an *swg-query* consists of a pattern over an alphabet of variables and types, a global window size and a tuple of local gap-size constraints. Syntactically, *swg-queries* are so-called Angluin-style patterns with variables, but with a semantics adapted to sequence data: each variable in the query string ranges only over a


AMW'23: 15th Alberto Mendelzon International Workshop on Foundations of Data Management, May 22–26, 2023, Santiago, Chile

[†]These authors contributed equally.

✉ andre.frochaux@informatik.hu-berlin.de (A. Frochaux); kleemeis@informatik.hu-berlin.de (S. Kleest-Meißner)

🆔 0009-0001-7918-8725 (A. Frochaux); 0000-0002-4133-7975 (S. Kleest-Meißner)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

single symbol and the query matches if, after replacing the variables by single data items, it occurs as a subsequence that satisfies the window size and local gap-size constraints. Angluin-style patterns were introduced in [7] and play a central role for inductive inference, in formal language theory and combinatorics on words (see [8], [9], [10]). Concepts and algorithms from inductive inference of so-called pattern languages, that can be described by Angluin-style patterns, can be adapted to swg-queries. Especially the notion of *descriptive patterns* (already introduced in [7], see also [11], [12]) forms a key concept and enables the adaptation of *Shinohara's algorithm* [13] for Angluin-style patterns. This algorithm computes a descriptive Angluin-style pattern upon input of a finite set of sequences of data items. The corresponding adaptation to swg-queries including some extensions were presented in [1], and lifted to a multi-dimensional data model in [14].

Subsequences in general have extensively been studied both in a purely combinatorial sense (in formal language theory, logic and combinatorics on words) and algorithmically (in string algorithms and bioinformatics); see the introductions of the recent papers [15], [16] for a comprehensive list of relevant pointers. The problem of matching subsequences with gap-constraints (and analysis problems with respect to the set of all gap-constrained subsequences of given strings) has been investigated in the recent papers [17], [18] (see also [19] for a survey).

Queries defined for complex event recognition (*CER*, for short) usually use operators such as sequencing, conjunction and disjunction, Kleene closure, negation and variables which may be bound to data items in a stream [20]. Inspired by the generalised gap-size constraints described in [17] that are defined over strings other than patterns over variables and types, and the use of disjunction in CER languages, we introduce two new notions of subsequence-queries, which both extend the expressive power of swg-queries in different ways:

- *disjunctive subsequence-queries with wildcards and local gap-size constraints*, for short: *dswg-queries*, and
- *subsequence-queries with wildcards and generalised gap-size constraints*, for short: *swgg-queries*

Improving the expressive power of the underlying language used for an automatically discovered descriptive query leads to results of increased precision. Enabling disjunction is a natural and effective way to reach this. Our second approach of generalised gap-size constraints allows detecting temporal contexts not only between consecutive data items, but between any data items in the query string.

The remainder of this paper is structured as follows. Section 2 introduces both, swgg-queries and dswg-queries, and discusses the relation to swg-queries. In Section 3, we provide a solution for the query discovery problem for both kinds of queries. Section 4 concludes the paper. Due to space limitations, proof details had to be deferred to the paper's full version [21].

2. Traces and Queries

This section introduces the syntax and semantics of both, swgg-query (Section 2.1), and dswg-queries (Section 2.2). For a better understanding we consider each extension individually.

By \mathbb{Z} , \mathbb{N} , $\mathbb{N}_{\geq 1}$ we denote the set of integers, non-negative integers, and positive integers, respectively. For every set M we denote the powerset by $\mathcal{P}(M)$, i.e. the set of all subsets

of M , and $\mathcal{P}_{\text{fin}}(M) := \{X \in \mathcal{P}(M) : X \text{ is finite}\}$ is the set of all finite subsets from M . Moreover, we write $\mathcal{P}_{\text{fin}}^+(M)$ for $(\mathcal{P}_{\text{fin}}(M) \setminus \emptyset)$ and for every $k \in \mathbb{N}_{\geq 1}$, we define $\mathcal{P}_k(M) := \{m \in \mathcal{P}_{\text{fin}}(M) \mid |m| = k\}$. For $\ell \in \mathbb{N}$ we let $[\ell] = \{i \in \mathbb{N}_{\geq 1} : 1 \leq i \leq \ell\}$. For a non-empty set A we write A^* (and A^+) for the set of all (non-empty) strings built from symbols in A . By $|s|$ we denote the length of a string s , and for a position $i \in [|s|]$ we write $s[i]$ to denote the letter at position i in s . A *factor* of a string $s \in A^*$ is a string $t \in A^*$ such that s is of the form $s_1 t s_2$ for $s_1, s_2 \in A^*$.

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

We model traces as finite, non-empty strings over some (finite or infinite) alphabet Γ of *types*. It will be reasonable to assume that $|\Gamma| \geq 2$. A *trace* (over Γ) is a string $t \in \Gamma^+$. We write $\text{types}(t)$ for the set of types that occur in t . Finally, we fix a countably infinite set Vars of *variables*, and we will always assume that Vars is disjoint with the set Γ of considered types.

2.1. Syntax and semantics of swgg-queries

Definition 1. An swgg-query $q = (s, w, C)$ (over Vars and Γ) is specified by

- a query string $s \in (\text{Vars} \cup \Gamma)^+$,
- a global window size $w \in \mathbb{N}_{\geq 1} \cup \{\infty\}$ with $w \geq |s|$ and
- a finite set C of generalised gap-size constraints (for $|s|$ and w) of form

$$(c^-, c^+, r)_j \in (\mathbb{N} \times \mathbb{N} \cup \{\infty\} \times \mathbb{N}_{\geq 1})_{\mathbb{N}_{\geq 1}}$$

for $j \in [|s| - 1]$, $r \leq c^- \leq c^+$ and $j + r \leq |s|$.

The semantics of swgg-queries is defined as follows: each variable in s represents an arbitrary type from Γ . A query $q = (s, w, C)$ *matches in a trace t* (in symbols: $t \models q$), if the wildcards in s can be replaced by types in Γ in such a way that the resulting string s' satisfies the following: t contains a factor t' of length at most w such that s' occurs as a subsequence in t' and for each $(c^-, c^+, r)_j \in C$ the gap between $s[j]$ and $s[j + r]$ in t' has length at least c^- and at most c^+ . Gaps of range r for $r \in [w]$ without any constraints are implicitly set to the most general constraint $(0, \infty, r)$.

A more formal description of these semantics relies on the following additional notation: An embedding $e : [\ell] \rightarrow [n]$ *satisfies a global window size w* , if $e(\ell) - e(1) + 1 \leq w$. Furthermore, it *satisfies a set of generalised gap-size constraints C* (for $|s|$ and w) if $c^- \leq e(j+r) - 1 - e(j) \leq c^+$, for each $(c^-, c^+, r)_j \in C$.

A *substitution* is a mapping $\mu : (\text{Vars} \cup \Gamma) \rightarrow (\text{Vars} \cup \Gamma)$ with $\mu(\gamma) = \gamma$ for all $\gamma \in \Gamma$. We lift substitutions to mappings $(\text{Vars} \cup \Gamma)^+ \rightarrow (\text{Vars} \cup \Gamma)^+$ in the obvious way, i.e. $\mu(s) = \mu(s[1])\mu(s[2]) \dots \mu(s[\ell])$ for $s \in (\text{Vars} \cup \Gamma)^+$ and $\ell := |s|$.

An swgg-query $q = (s, w, C)$ *matches in a trace $t \in \Gamma^+$* (or *t matches q*), if and only if there are a substitution $\mu : (\text{Vars} \cup \Gamma) \rightarrow \Gamma$ and an embedding $e : [\ell] \rightarrow [n]$ that satisfies w and C , such that $\mu(s) \preceq_e t$. We call (μ, e) a *witness* for $t \models q$.

The *model set* of a query q w.r.t. to a type set $\Delta \subseteq \Gamma$ is $\text{Mod}_\Delta(q) := \{t \in \Delta^+ : t \models q\}$. Note that there exist swgg-queries q such that $\text{Mod}_\Gamma(q) = \emptyset$. They have in common that either their generalised gap-size constraints conflict with the global window size or some gap-size constraints are in conflict among themselves. Lemma 3 characterises swgg-queries with compatible constraints.

Example 2. Let $\Gamma = \{a, b\}$. Let $q_1 = (s, w, C_1)$ and $q_2 = (s, w, C_2)$ be the two swgg-queries over Vars, where $s = a a a a a$ and $w = 10$ for both queries, and $C_1 = ((7, 7, 3)_1, (6, 6, 3)_2, (0, 0, 1)_5)$ and $C_2 = ((4, 4, 5)_1, (2, 5, 2)_3)$.

$$\begin{array}{cccccccccccccccc}
& & 1 & 2 & 3 & 4 & 5 & & 6 & & & 1 & 2 & 3 & 4 & 5 & 6 \\
s & = & a & a & a & a & a & & a & & s & = & a & a & a & a & a & a \\
C_1 & : & & \underbrace{\hspace{2cm}} & & & \underbrace{\hspace{2cm}} & & & & C_2 & : & & \underbrace{\hspace{2cm}} & & & \\
& & & (7, 7, 3)_1 & & & (0, 0, 1)_5 & & & & & & & (2, 5, 2)_3 & & & \\
& & & \underbrace{\hspace{2cm}} & & & & & & & & & & \underbrace{\hspace{2cm}} & & & \\
& & & (6, 6, 3)_2 & & & & & & & & & & (4, 4, 5)_1 & & & \\
w & : & & \underbrace{\hspace{4cm}} & & & & & & & w & : & & \underbrace{\hspace{4cm}} & & & \\
& & & & & & 10 & & & & & & & & & & 10
\end{array}$$

A shortest trace over Γ which satisfies C_1 is $t = a b a b b b b a a a$. But $t \not\models q_1$ since t does not satisfy $w = 10$. Since t is a shortest trace there exists no trace satisfying both, w and C . (The shortest trace is not unique since the sequences of bs could be replaced by arbitrary types from Γ .)

Note that $\text{Mod}_\Gamma(q_2) = \emptyset$ holds as well since $(4, 4, 5)_1$ and $(2, 5, 3)_3$ are incompatible: For each trace t , substitution μ and embedding e such that $\mu(s) \preceq_e t$ and e satisfies $(4, 4, 5)_1$, the second gap-size constraint is not satisfied, since it demands at least one further type between $e(3)$ and $e(4)$, contradicting $(4, 4, 5)_1$.

Lemma 3. An swgg-query $q = (s, w, C)$ (over Vars and Γ) is satisfiable, i.e. $\text{Mod}_\Gamma(q) \neq \emptyset$, iff there are no two sequences

$$C' = \left((c_1^-, c_1^+, r_1')_{j_1'}, (c_2^-, c_2^+, r_2')_{j_2'}, \dots, (c_{|C'|}^-, c_{|C'|}^+, r_{|C'|}')_{j_{|C'|}' } \right)$$

and

$$C'' = \left((c_1''^-, c_1''^+, r_1'')_{j_1''}, (c_2''^-, c_2''^+, r_2'')_{j_2''}, \dots, (c_{|C''|}''^-, c_{|C''|}''^+, r_{|C''|}'')_{j_{|C''|}'' } \right)$$

from $C \cup \{(0, \infty, 1)_1, \dots, (0, \infty, 1)_{|s|-1}\}$ where $j_1' = j_1''$, $j_{|C'|}' + r_{|C'|}' = j_{|C''|}'' + r_{|C''|}''$, and $j_{i+1}' = j_i' + r_i'$ and $j_{i+1}'' = j_i'' + r_i''$ for all $i \in [|C'| - 1]$, with

- (i) $|s| + \sum_{i=1}^{|C'|} c_i^- - r_i' + 1 > w$, or
- (ii) $\sum_{i=1}^{|C'|} c_i^- - r_i' + 1 > \sum_{i=1}^{|C''|} c_i''^+ - r_i'' + 1$.

For the rest of this paper, we only focus on queries with a non-empty model set.

2.2. Syntax and semantics of dswg-queries

Definition 4. A dswg-query $q = (s, w, c)$ (over Vars and Γ) is specified by

- a query string $s = s[1] \dots s[\ell]$, whereby $s[i] = \begin{cases} x \in \text{Vars} \\ \chi \in \mathcal{P}_{\text{fin}}^+(\Gamma) \end{cases}$,
- a global window size $w \in \mathbb{N}_{\geq 1} \cup \{\infty\}$ with $w \geq |s|$ and
- a tuple $c = (c_1, c_2, \dots, c_{|s|-1})$ of local gap-size constraints (for $|s|$ and w), where $c_i = (c_i^-, c_i^+) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$, such that $c_i^- \leq c_i^+$ for every $i \in [|s|-1]$ and $|s| + \sum_{i=1}^{|s|} c_i^- \leq w$.

Note that setting all gap-size constraints of a dswg-query q to $(0, \infty)$ corresponds to a query without gap-size constraints.

The semantics of dswg-queries is defined as follows: Again, variables in s represent an arbitrary type, and each set χ stands for a disjunction. Intuitively, a trace t matches a query $q = (s, w, c)$ if the variables in s can be replaced by types and each occuring of an χ can be mapped to a single type $\gamma \in \chi$, such that the resulting string s' occurs as a subsequence in t that spans at most w types and the gap between $s'[i]$ and $s'[i+1]$ in t has length between c_i^- and c_i^+ , for all $i < \ell := |s|$.

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

A substitution of size ℓ is a mapping $\mu_\ell : ([\ell] \times \text{Vars} \cup \mathcal{P}_{\text{fin}}^+(\Gamma)) \rightarrow (\text{Vars} \cup \mathcal{P}_{\text{fin}}^+(\Gamma))$ with:

$$\mu_\ell(i, z) = \begin{cases} x \in (\text{Vars} \cup \mathcal{P}_{\text{fin}}^+(\Gamma)) & , i = 1 \text{ and } z \in \text{Vars} \\ \mu_\ell(1, z) & , i > 1 \text{ and } X \in \text{Vars} \\ z' & , \emptyset \subset z' \subseteq z \text{ for all non-empty } z \subseteq_{\text{fin}} \Gamma. \end{cases}$$

We extend substitutions of size ℓ to mappings $([\ell] \times \text{Vars} \cup \mathcal{P}_{\text{fin}}^+(\Gamma))^+ \rightarrow (\text{Vars} \cup \mathcal{P}_{\text{fin}}^+(\Gamma))^+$ for strings $s \in (\text{Vars} \cup \mathcal{P}_{\text{fin}}^+(\Gamma))^+$ of size ℓ in the obvious way, i.e., $\mu(s) = \mu_\ell(1, s[1])\mu_\ell(2, s[2]) \dots \mu_\ell(\ell, s[\ell])$. Since the size of the string must match the size of the substitution, we can omit the index ℓ if we apply it to a string. Particularly, we can omit the parameter i for the position if the second parameter is a variable, as we have for all variables z that $\mu(i, z) = \mu(i', z)$ for all $i, i' \in [\ell]$, or if the position is given by the context, i.e. we write $\mu(s[i])$ instead of $\mu(i, s[i])$.

A dswg-query $q = (s, w, c)$ matches in a trace $t \in \Gamma^+$ (or, t matches q , in symbols: $t \models q$), if and only if there are a substitution $\mu : ([\ell] \times \text{Vars} \cup \mathcal{P}_{\text{fin}}^+(\Delta)) \rightarrow \Gamma$ (i.e., there are only singletons in the co-domain of μ and every type of $\mu(s)$ is the unique element of its singleton) and an embedding $e : [|s|] \rightarrow [|t|]$ that satisfies w and c , such that $\mu(s) \preceq_e t$. We call (μ, e) a witness for $t \models q$.

Example 5. Let $x_1, x_2, x_3 \in \text{Vars}$ and $\Gamma = \{a, b, c\}$. We consider a query $q = (s, w, c)$, where $s = x_1\{a, b\}x_1x_2\{c\}x_3\{a, c\}x_1$, $w = 25$ and $c = ((0, 1), (2, \infty), (3, \infty), (0, 5), (0, 5), (1, 5), (1, 2))$. For $t_1, t_2 \in \Gamma^*$ we consider the trace $t = t_1 c a b b c a b a c a b a c b c b b a c t_2$. We observe that $t \models q$, and a witness substitution and embedding can be illustrated as follows:

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

We close this subsection with two little observations. First, it is reasonable to assume that χ is a proper subset of Γ , otherwise we could also use a wildcard instead of a disjunction. Second, in the case of a finite alphabet Γ , we obtain the possibility to express a simple kind of negation. We can build a query where s contains a substring $s' = a \chi b$ with $\chi = \Gamma \setminus \{c\}$ and corresponding conditions $c = ((0, 0), (0, 0))$ that is only matched by traces where in between of the relevant a and b occurs exactly one letter that is not c . Unfortunately, it is not possible to express negation in general, so we cannot express the following: a and b have one or two letters in between, none is c .

2.3. About containment

This section is dedicated to a characterisation of containment, a classical property considered in database theory.

An swgg-query q is called an (ℓ, w, C) -swgg-query (over Vars and Γ) if $q = (s, w, C)$ with $|s| = \ell$, (ℓ, w, c) -dswg-queries are analogously defined. The parameter ℓ will be called *string length*. If the maximal size of typesets occurring in an (ℓ, w, c) -dswg-query q is bounded by a number $k \geq 1$, we call q an (ℓ, w, c, k) -dswg-query (for swgg-queries k always equals 1).

Given an swgg-query or dswg-query q we omit the prefix and call q a *query*, if q may be both, or it is clear from the context whether q is an swgg-query or dswg-query. We use (ℓ, w, \tilde{c}) -query (or (ℓ, w, \tilde{c}, k) -query) as notation for queries which might be swgg-query or dswg-query and assume that the gap-size constraints \tilde{c} are compatible with ℓ and w or satisfy Lemma 3, respectively.

We write $\text{types}(q)$ (or $\text{types}(s)$), $\text{typesets}(q)$ (or $\text{typesets}(s)$) and $\text{vars}(q)$ (or $\text{vars}(s)$) for the set of types, the set of all typesets and the set of variables, respectively, that occur in q 's query string s . I.e.,

$$\begin{aligned} \text{types}(q) &:= \{\gamma \in \Gamma \mid \text{there ex. } i \in [|s|] : \gamma \in s[i]\} \\ \text{typesets}(q) &:= \{\chi \subseteq \Gamma \mid \text{there ex. } i \in [|s|] : s[i] = \chi\} \\ \text{vars}(q) &:= \{x \in \text{Vars} \mid \text{there ex. } i \in [|s|] : s[i] = x\} \end{aligned}$$

For the reason of readability we omit braces in query strings if the set consists only of a unique element. Therefore, we write for example $s = x_1 a b x_2 a \{a, c\} b$ instead of $s = x_1 \{a\} \{b\} x_2 \{a\} \{a, c\} \{b\}$. Vice versa, we can consider a query string s over $\text{Vars} \cup \Gamma$ as a string over $\text{Vars} \cup \mathcal{P}_1(\Gamma)$ where every $s[i]$ is a singleton.

A query q is said to be *contained* in a query q' w.r.t. to a set $\Delta \subseteq \Gamma$ (we write $q \subseteq_{\Delta} q'$) if $\text{Mod}_{\Delta}(q) \subseteq \text{Mod}_{\Delta}(q')$.

Definition 6. A homomorphism from q' to q is a substitution h such that $h(s') = s$ and the following property holds:

For every $z \in \text{Vars}$ that occurs at least twice in the query string s' of q' and is mapped to a subset of Γ via h , we have $h(z)$ is a singleton.

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

This additional property of homomorphisms feels arbitrary or artificial, but it is perfectly tailored to our discovery algorithm and if the considered class of queries in Definition 6 is the class of all (ℓ, w, C) -swgg-query, then in any way, we have $s[i]$ is a singleton for all $i \in [\ell]$. Now, the following theorem gives a characterisation of containment.

Theorem 7. *Given some sufficiently large Γ . Let q and q' be (s, w, \tilde{c}) -queries over Vars and Γ . If q and q' are satisfiable, it holds, that:*

$$q \subseteq_{\Gamma} q' \iff q' \xrightarrow{\text{hom}} q.$$

Sufficiently large, in the context of Theorem 7 means, $|\Gamma| \geq 2$ for the case of dswg-queries. In the case of swgg-queries the size of Γ depends on gap-size constraints with range greater than 1. Intuitively, the necessary size of Γ depicts how much structure information of q' can be hidden in the gaps of q . For further information and an example consider the theorems proof in the paper's full version [21].

2.4. Correlation to swg-queries

Note that an swgg-query query $q = (s, w, C)$ with $C = \{c_1, \dots, c_{\ell-1}\}$ and $r_i = 1$ for all $i \in [\ell - 1]$ precisely corresponds to the notion of swg-queries introduced in [1]. Let q be an (ℓ, w, c, k) -query containing typesets over $\mathcal{P}_{\text{fin}}^+(\Gamma)$. If $k = 1$ this corresponds to the syntax and semantics of swg-queries as well.

In [14] a mapping between one-dimensional and multi-dimensional sequence data was introduced, such that a multi-dimensional trace matches a multi-dimensional query if and only if the corresponding one-dimensional trace matches the corresponding one-dimensional query. This mapping can be adapted to swgg-queries and dswg-queries.

3. Discovery

The question of how meaningful swgg-queries and dswg-queries can be discovered from a given set of traces is of peculiar interest and was answered algorithmically in [1] for swg-queries. We adapt these results to swgg-queries and dswg-queries.

A *sample* is a finite, non-empty set \mathcal{S} of traces over Γ . Given a sample \mathcal{S} , let $\Gamma_{\mathcal{S}}$ be the set of all types occurring in \mathcal{S} , i.e. $\bigcup_{t \in \mathcal{S}} \text{types}(t)$. The *support* $\text{supp}(q, \mathcal{S})$ of a query q in \mathcal{S} is defined as the fraction of traces in the sample that match q , i.e. $\text{supp}(q, \mathcal{S}) := \frac{|\{t \in \mathcal{S} : t \models q\}|}{|\mathcal{S}|}$. A *support threshold* is a rational number sp with $0 < \text{sp} \leq 1$. A query q is said to *cover* a sample \mathcal{S} with *support* sp if $\text{supp}(q, \mathcal{S}) \geq \text{sp}$. Let \mathcal{S} be a sample, sp be a support threshold and $k \in [|\Gamma_{\mathcal{S}}| - 1]$. An (ℓ, w, \tilde{c}, k) -query q is called *descriptive for \mathcal{S} w.r.t. $(\text{sp}, (\ell, w, \tilde{c}, k))$* if $\text{supp}(q, \mathcal{S}) \geq \text{sp}$, and there is no other (ℓ, w, \tilde{c}, k) -query q' with $\text{supp}(q', \mathcal{S}) \geq \text{sp}$ and $q' \subset_{\Gamma} q$. A type $\gamma \in \Gamma$ (or a typeset $\chi \in \mathcal{P}_{\text{fin}}^+(\Gamma)$) *satisfies* sp w.r.t. to \mathcal{S} , if the fraction of traces containing γ (or some $\gamma \in \chi$) is greater than or equal to sp . The set of all types (or typesets) that satisfy sp w.r.t. to \mathcal{S} is

$$\Delta(\mathcal{S}, \text{sp}) := \{\chi \in \mathcal{P}_{\text{fin}}^+(\Gamma) : \frac{|\{t \in \mathcal{S} : \text{ex. } \gamma \in \chi \text{ s.t. } \gamma \in \text{types}(t)\}|}{|\mathcal{S}|} \geq \text{sp}\}.$$

We omit \mathcal{S} and sp , if they are clear from the context. For $i \in [|\Gamma_{\mathcal{S}}| - 1]$ we write Δ_i to denote the subset of Δ which contains only typesets of size i . For a descriptive (ℓ, w, \tilde{c}, k) -query q , $\text{typesets}(q) \subseteq \Delta_1 \dot{\cup} \dots \dot{\cup} \Delta_k$ holds. This corresponds to $\Delta = \Delta_1 = \{\gamma \in \Gamma : \frac{|\{t \in \mathcal{S} : \gamma \in \text{types}(t)\}|}{|\mathcal{S}|} \geq \text{sp}\}$ if the considered query is an swgg-query.

Given an (ℓ, w, \tilde{c}, k) -query $q = (s, w, \tilde{c})$ and a symbol z from $\text{Vars} \cup \mathcal{P}_{\text{fin}}^+(\Gamma)$ we let $\text{pos}(q, z) = \text{pos}(s, z) = \{i \in [\ell] : s[i] = z\}$ be set of all positions i in s that carry z . Given a set of positions $P \subseteq [\ell]$, and a symbol $z \in \text{Vars} \cup \mathcal{P}_{\text{fin}}^+(\Gamma)$ we write $s\langle P \mapsto z \rangle$ to denote the query string s' which is obtained from s by setting $s[i]$ to z , for all $i \in P$. Let $x \in \text{Vars}$. We write $s\langle x \mapsto z \rangle$ as an abbreviation for $s\langle \text{pos}(q, x) \mapsto z \rangle$. Next, we present the algorithmical idea for query discovery:

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

Pseudocode of an algorithm solving CompDescQuery is provided in Algorithm 1. Given \mathcal{S} , sp and query parameters (ℓ, w, \tilde{c}, k) as input, the algorithm first builds the *most general query* $q = q_{\text{mg}}$ for (ℓ, w, \tilde{c}, k) . Its query string consists of ℓ distinct variables, i.e. $s_{\text{mg}} = x_1 \dots x_\ell$, and q_{mg} is most general in the sense that $q' \subseteq_{\Gamma} q_{\text{mg}}$ for each (ℓ, w, \tilde{c}, k) -query q' . If $\text{supp}(q, \mathcal{S}) < \text{sp}$

ALGORITHM 1: DescrQuery($\mathcal{S}, \text{sp}, (\ell, w, \tilde{c}, k)$)

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

- 1 $s := s_{\text{mg}}; q := (s_{\text{mg}}, w, \tilde{c})$ // query string and query
- 2 **if** $\text{supp}(q, \mathcal{S}) < \text{sp}$ **then stop and return** \perp
- 3 $\Delta := \Delta_1 \dot{\cup} \dots \dot{\cup} \Delta_k$ // typesets to be considered
- 4 $U := \text{vars}(q); V := \emptyset$ // unvisited and available variables
- 5 **while** $U \neq \emptyset$ **do**
- 6 **select** an arbitrary $x \in U$ and let $U := U \setminus \{x\}$ and $\Delta_1 := \Delta_1 \cup V$
- 7 **for** $i = 1$ **to** k **do**
- 8 $\text{replace} := \text{False}$
- 9 **while** $\Delta_i \neq \emptyset$ **do**
- 10 **select** an arbitrary $y \in \Delta_i$ and let $\Delta_i := \Delta_i \setminus \{y\}$
- 11 $q' := (s\langle x \mapsto y \rangle, w, \tilde{c})$
- 12 **if** $\text{supp}(q', \mathcal{S}) \geq \text{sp}$ **then**
- 13 $s := s\langle x \mapsto y \rangle; \text{replace} := \text{True}$ // ReplaceOp
- 14 **break for loop**
- 15 **if** replace is **False** **then** $V := V \cup \{x\}$ // NoChangeOp
- 16 **stop and return** $q := (s, w, \tilde{c})$

the algorithm stops and returns \perp (line 2), because no other query q' with $q' \subseteq_{\Gamma} q = q_{\text{mg}}$ can describe \mathcal{S} with sufficient support.

Otherwise, the algorithm searches for an admissible *replacement operation* for each variable $x \in U := \text{vars}(s) = \{x_1, \dots, x_\ell\}$ during the main loop (Line 5). A replacement operation

replaces x by an element $y \in \Delta_i$ (during the i -th iteration of the for-loop) which may be a typeset or an *available* variable $y \in V$ (if $i=1$). The replacement operation is stored in q and called *admissible* if the resulting query satisfies the support threshold (lines 11–13). If $\text{supp}(\langle \text{pos}(q, x) \mapsto y \rangle, \mathcal{S}) < \text{sp}$ for all $y \in \Delta_i$ and all $i \in [k]$ the query string remains unchanged and x gets available (line 15). After each variable in $\text{vars}(s_{\text{mg}})$ has been considered, the algorithm terminates and produces the current query as output (line 16).

Next we depict an exemplaric run of Algorithm 1. We refer to the paper’s full version [21] for a brief discussion, why Δ is passed through incrementally in Line 7.

Example 8. Let $\Gamma = \{a, b, c\}$ and $x_1, x_2, x_3 \in \text{Vars}$. Consider the sample $\mathcal{S} = \{a b b, a c c\}$, $\text{sp} = 1.0$, $\ell = w = 3$, $c = ((0, 0), (0, 0))$ and $k = 2$.

On input $(\mathcal{S}, \text{sp}, (\ell, w, c, k))$ the algorithm first generates $q = (x_1 x_2 x_3, w, c)$. Since q satisfies the support threshold the algorithm proceeds by computing $\Delta = \{\{a\}\} \cup \{\{a, b\}, \{a, c\}, \{b, c\}\}$. Assume the algorithm selects $x := x_3$ during the first iteration of the main loop. It turns out that $\Delta_1 = \{\{a\}\}$ does not contain a typeset for an admissible replacement of x_3 . Hence, the algorithm considers Δ_2 during the second transition of the for-loop in Line 7. The only admissible replacement is $s \langle x_3 \mapsto \{b, c\} \rangle$, and s is replaced by $x_1 x_2 \{b, c\}$ (V remains empty).

Let us assume that during the second transition through the main loop the algorithm selects $x := x_1$ and $y := \{a\} \in \Delta_1$. The replacement of x_1 by $\{a\}$ is admissible (as it has support 1 on \mathcal{S}). Therefore, s is replaced by $\{a\} x_2 \{b, c\}$ and V remains unchanged again.

In its last iteration (during the second transition through the for-loop), $s \langle x_2 \mapsto \{b, c\} \rangle$ is the only admissible replacement operation. The run terminates after this iteration and outputs the query $q = (s, w, c)$ with $s = \{a\} \{b, c\} \{b, c\}$.

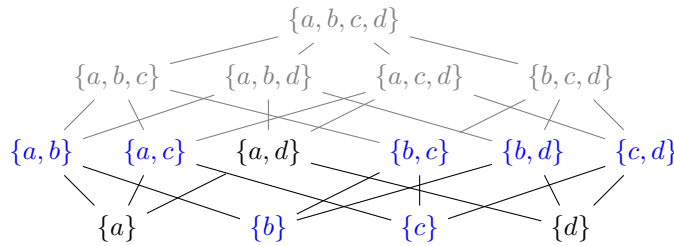


Figure 1: Let $\Gamma = \{a, b, c, d\}$, $\mathcal{S} = \{c a b b c a c b, c b b b a c c b, c c b b c c c b\}$, $\text{sp} = 1.0$ and $k = 2$. Depicted is a top-down walk through $\mathcal{P}_{\text{fin}}^+(\Gamma)$ for \mathcal{S} , starting with typesets of size $k = 2$. The typesets marked in blue represent Δ .

Note that algorithm 1 computes a swgg-query if $k = 1$ and generalised gap-size constraints are given. Furthermore, during each iteration of the main loop, the for-loop is transisted only once. It remains to discuss how Δ can be calculated in case that $k > 1$. Starting with all subsets χ of $\mathcal{P}_{\text{fin}}^+(\Gamma)$ with $|\chi| = k$ it suffices to explore $\mathcal{P}_{\text{fin}}^+(\Gamma)$ in a top-down manner: we walk through the search space level-wise and check whether the current typesets satisfy sp w.r.t. \mathcal{S} . If this is not the case for a typeset χ , all typesets $\chi' \subset \chi$ can be deleted from the search space, since they do not satisfy sp . An example is depicted in Figure 1.

Theorem 9. Given some sufficiently large Γ . Let \mathcal{S} be a sample, let sp be a support threshold with $0 < \text{sp} \leq 1$, let (ℓ, w, \tilde{c}, k) be query parameters with $k = 1$ if $\tilde{c} = C$.

- (a) If there does not exist any (ℓ, w, \tilde{c}, k) -swgg- or dswg-query that is descriptive for \mathcal{S} w.r.t. $(\text{sp}, (\ell, w, \tilde{c}, k))$ 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 $(\mathcal{S}, \text{sp}, (\ell, w, \tilde{c}, k))$ terminates and outputs an swgg-query or dswg-query q (depending on k), with $|\chi| \leq k$ for all $\chi \in \text{typesets}(q)$, that is descriptive for \mathcal{S} w.r.t. $(\text{sp}, (\ell, w, \tilde{c}, k))$.

We refer to the paper’s full version [21] for the proof. Analysing the complexity of the algorithm, identifies two bottle necks. First the Δ -calculation in the case of dswg-queries. This can be handled by adjusting the parameter k , i.e. by bounding the size of the disjunctive clauses in the query string. The second is already known from [1] 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 [1]. Membership can be obtained by guessing a witness.

4. Conclusion and Future Work

We model sequence data as traces and discover descriptive queries over traces to find a characteristic template for situations of interests. Since an increased expressive power of the underlying query language leads to a more detailed picture of sois, we extended swg-queries, introduced in [1], in two different ways. First, by generalising the gap size constraints (Section 2.1) and second, by adding the possibility of disjunctions (Section 2.2). We adopted and extended the discovery algorithm to our approach and ensured that the essential complexity properties are preserved (Section 3). Note that the extended approach can be applied to the multi-dimensional setting, analogously to [14].

For future work we will merge both extensions to one query language. We are interested in a more general notion of disjunction and negation, and a more generous possibility to describe gaps. For the latter [17] is a good yardstick. An in-depth (parameterised) complexity analysis is intended as well. Since the crucial point is the inherent complexity of the matching problem, we are working on data structures to improve the computation in practical application. In the long run we will investigate containment for relaxed query parameters (ℓ, w, c, k) .

Acknowledgments

We thank Markus L. Schmid for useful discussions. Sarah Kleest-Meißner was supported by the German Research Foundation (DFG), CRC 1404: “FONDA: Foundation of Workflows for Large-Scale Scientific Data Analysis”.

References

- [1] S. Kleest-Meißner, R. Sattler, M. L. Schmid, N. Schweikardt, M. Weidlich, Discovering event queries from traces: Laying foundations for subsequence-queries with wildcards and gap-size constraints, in: 25th International Conference on Database Theory, ICDT 2022, volume 220 of *LIPICs*, 2022, pp. 18:1–18:21. doi:10.4230/LIPICs.ICDT.2022.18.
- [2] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at google with borg, in: L. Réveillère, T. Harris, M. Herlihy (Eds.), Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21–24, 2015, ACM, 2015, pp. 18:1–18:17. URL: <https://doi.org/10.1145/2741948.2741964>. doi:10.1145/2741948.2741964.
- [3] A. Artikis, M. Weidlich, F. Schnitzler, I. Boutsis, T. Liebig, N. Piatkowski, C. Bockermann, K. Morik, V. Kalogeraki, J. Marecek, A. Gal, S. Mannor, D. Gunopulos, D. Kinane, Heterogeneous stream processing and crowdsourcing for urban traffic management, in: S. Amer-Yahia, V. Christophides, A. Kementsietsidis, M. N. Garofalakis, S. Idreos, V. Leroy (Eds.), Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24–28, 2014, OpenProceedings.org, 2014, pp. 712–723. URL: <https://doi.org/10.5441/002/edbt.2014.77>. doi:10.5441/002/edbt.2014.77.
- [4] K. Teymourian, M. Rohde, A. 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.
- [5] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: L. Popa, S. Abiteboul, P. G. Kolaitis (Eds.), Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3–5, Madison, Wisconsin, USA, ACM, 2002, pp. 1–16. URL: <https://doi.org/10.1145/543613.543615>. doi:10.1145/543613.543615.
- [6] A. Artikis, C. Baber, P. Bizarro, C. Canudas-de-Wit, O. Etzion, F. Fournier, P. Goulart, A. Howes, J. Lygeros, G. Paliouras, A. Schuster, I. Sharfman, Scalable proactive event-driven decision making, *IEEE Technol. Soc. Mag.* 33 (2014) 35–41. URL: <https://doi.org/10.1109/MTS.2014.2345131>. doi:10.1109/MTS.2014.2345131.
- [7] D. Angluin, Inductive inference of formal languages from positive data, *Inf. Control.* 45 (1980) 117–135. URL: [https://doi.org/10.1016/S0019-9958\(80\)90285-5](https://doi.org/10.1016/S0019-9958(80)90285-5). doi:10.1016/S0019-9958(80)90285-5.
- [8] T. Shinohara, S. 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.
- [9] F. Manea, M. L. Schmid, Matching patterns with variables, in: R. Mercas, D. Reidenbach (Eds.), Combinatorics on Words - 12th International Conference, WORDS 2019, Loughborough, UK, September 9–13, 2019, Proceedings, volume 11682 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 1–27. URL: https://doi.org/10.1007/978-3-030-28796-2_1. doi:10.1007/978-3-030-28796-2_1.
- [10] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Volume 1: Word, Language, Grammar, Springer, 1997. URL: <https://doi.org/10.1007/978-3-642-59136-5>. doi:10.1007/978-3-642-59136-5.

- [11] D. D. Freydenberger, D. Reidenbach, Existence and nonexistence of descriptive patterns, *Theor. Comput. Sci.* 411 (2010) 3274–3286. URL: <https://doi.org/10.1016/j.tcs.2010.05.033>. doi:10.1016/j.tcs.2010.05.033.
- [12] D. D. Freydenberger, D. Reidenbach, Inferring descriptive generalisations of formal languages, *J. Comput. Syst. Sci.* 79 (2013) 622–639. URL: <https://doi.org/10.1016/j.jcss.2012.10.001>. doi:10.1016/j.jcss.2012.10.001.
- [13] T. Shinohara, Polynomial time inference of pattern languages and its application, in: *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science, MFCS, 1982*, pp. 191–209.
- [14] S. Kleest-Meißner, R. Sattler, M. L. Schmid, N. Schweikardt, M. Weidlich, Discovering multi-dimensional subsequence queries from traces - from theory to practice, in: B. König-Ries, S. Scherzinger, W. Lehner, G. Vossen (Eds.), *Datenbanksysteme für Business, Technologie und Web (BTW 2023)*, 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS), 06.-10, März 2023, Dresden, Germany, *Proceedings*, volume P-331 of *LNI*, Gesellschaft für Informatik e.V., 2023, pp. 511–533. URL: <https://doi.org/10.18420/BTW2023-24>. doi:10.18420/BTW2023-24.
- [15] P. Gawrychowski, M. Kosche, T. Koß, F. Manea, S. Siemer, Efficiently testing simon’s congruence, in: M. Bläser, B. Monmege (Eds.), *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021*, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference), volume 187 of *LIPICs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 34:1–34:18. URL: <https://doi.org/10.4230/LIPICs.STACS.2021.34>. doi:10.4230/LIPICs.STACS.2021.34.
- [16] J. D. Day, P. Fleischmann, M. Kosche, T. Koß, F. Manea, S. Siemer, The edit distance to k -subsequence universality, in: M. Bläser, B. Monmege (Eds.), *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021*, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference), volume 187 of *LIPICs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 25:1–25:19. URL: <https://doi.org/10.4230/LIPICs.STACS.2021.25>. doi:10.4230/LIPICs.STACS.2021.25.
- [17] J. D. Day, M. Kosche, F. Manea, M. L. Schmid, Subsequences with gap constraints: Complexity bounds for matching and analysis problems, *CoRR* abs/2206.13896 (2022). URL: <https://doi.org/10.48550/arXiv.2206.13896>. doi:10.48550/arXiv.2206.13896. arXiv:2206.13896.
- [18] M. Kosche, T. Koß, F. Manea, V. Pak, Subsequences in bounded ranges: Matching and analysis problems, *CoRR* abs/2207.09201 (2022). URL: <https://doi.org/10.48550/arXiv.2207.09201>. doi:10.48550/arXiv.2207.09201. arXiv:2207.09201.
- [19] M. Kosche, T. Koß, F. Manea, S. Siemer, Combinatorial algorithms for subsequence matching: A survey, 2022. arXiv:2208.14722.
- [20] N. Giatrakos, E. Alevizos, A. Artikis, A. Deligiannakis, M. N. Garofalakis, Complex event recognition in the big data era: a survey, *VLDB J.* 29 (2020) 313–352. URL: <https://doi.org/10.1007/s00778-019-00557-w>. doi:10.1007/s00778-019-00557-w.
- [21] A. Frochaux, S. Kleest-Meißner, Puzzling over subsequence-query extensions: Disjunction and generalised gaps, *CoRR* 2305.08236 (2023). URL: <https://doi.org/10.48550/arXiv.2305.08236>. doi:10.48550/arXiv.2305.08236. arXiv:2305.08236.