Mining behavioral sequence constraints for classification

Mining Behavioral Sequence Constraints for Classification

Johannes De Smedt*, Galina Deeva†, and Jochen De Weerdt†

Abstract—Sequence classification deals with the task of finding discriminative and concise sequential patterns. To this purpose, many techniques have been proposed, which mainly resort to the use of partial orders to capture the underlying sequences in a database according to the labels. Partial orders, however, pose many limitations, especially on expressiveness, i.e. the aptitude towards capturing certain behavior, and on conciseness, i.e. doing so in a compact and informative way. These limitations can be addressed by using a better representation. In this paper we present the interesting Behavioral Constraint Miner (iBCM), a sequence classification technique that discovers patterns using behavioral constraint templates. The templates comprise a variety of constraints and can express patterns ranging from simple occurrence, to looping and position-based behavior over a sequence. Furthermore, iBCM also captures negative constraints, i.e. absence of particular behavior. The constraints can be discovered by using simple string operations in an efficient way. Finally, deriving the constraints with a window-based approach allows to pinpoint where the constraints hold in a string, and to detect whether patterns are subject to concept drift. Through empirical evaluation, it is shown that iBCM is better capable of classifying sequences more accurately and concisely in a scalable manner.

Index Terms—Sequence classification, sequential pattern mining, behavioral constraint templates, Declare.

1 Introduction

SEQUENCE mining has experienced a vast surge in interest in recent years, finding its application in numerous domains, such as bioinformatics [1], text mining [2], road analysis [3] and user behavior analysis [4]. More recently, it has also been proven useful for analyzing educational data [5].

There are various sequence classification techniques that can discriminate between classes of sequences by deriving sequential patterns from temporal databases, which in turn can vary depending on the discovery technique, e.g. prefix-oriented and constraint-based approaches, or in terms outcomes, e.g. regular expressions and closed sequences. These derived sequential patterns are then used for classifying new database entries, which requires to construct the most discriminating rather than the most complete set of features.

In our preliminary study [6], we introduced iBCM (interesting Behavioral Constraint Miner), a novel sequence classification technique that featurizes sequences according to a predefined set of behavioral constraint templates. An example template is alternate precedence(a,b), indicating that every occurrence of b needs to be preceded by a new occurrence of a. iBCM outputs a set of many such constraints over the items in the sequences per class (e.g. the document type). Together, the constraints form an image of what sequential patterns are present and different for each class. In this way, it is possible to obtain a more fine-granular view of the temporal relations between items, which in turn can be applied for classification.

This paper extends our previous work in four different ways. First, iBCM is further developed into a window-based pattern discovery approach, which allows for better capturing concept drift in a sequence database. When the generator of the sequences changes its behavior over time, this window-based approach is capable of recognizing this change. Secondly, we explicitly demonstrate that iBCM is more concise (less constraints to represent the same information), more expressive (a richer set of constraints, e.g. also negative constraints), and more scalable. Thirdly, it is shown how iBCM brings additional insight into the characteristics of sequential databases, by exploiting the information gain criterion at the level of constraint types to get (meta-level) insight into the characteristics that can lead to accurate classification of sequences. Finally, we present an extensive comparative experimental evaluation of iBCM w.r.t. alternative sequence classification techniques showing that the presented technique is best capable to obtain high discriminative power while minimizing the number of features needed, and this with the lowest computational complexity.

In summary, iBCM exhibits the following advantages:

- It employs a rich, varied set of behavioral constraint templates that can be derived in a fast manner;
- It can be extended to incorporate any regular expression;
- It includes negative constraints for providing counter evidence, useful towards classification;
- It includes both unary cardinalities, as well as relational constraints;
- It enables easy comparison of constraint sets;
- It enables understanding what type of behavioral relations are present;
- It is capable of pinpointing the behavior to a certain part of the string, and hence is also robust against concept drift, defined as a change in the underlying system that

* *Johannes De Smedt is with the University of Edinburgh, johannes.desmedt@ed.ac.uk*
† *Galina Deeva and Jochen De Weerdt are with KU Leuven, Belgium, {galina.deeva,jochen.deweerdt}@kuleuven.be*

generates the items;
- It can be converted into a global automaton for representing behavior graphically.

This paper is structured as follows. Section 2 discusses the state-of-the-art of both sequence mining and classification. Next, we present the backdrop for mining behavioral sequence patterns in Section 3, and consequently discuss the inference part of iBCM in Section 4. Next, Section 5 reports on an extensive comparative experimental evaluation with other state-of-the-art techniques. Finally, Section 6 summarizes the contributions and provides suggestions for future work.

2 STATE-OF-THE-ART

This section gives an overview of existing sequence mining and sequence classification techniques.

2.1 Sequence Mining

Sequence mining, also referred to as frequent ordered itemset mining or temporal data mining, has been widely studied in recent years. The initial approach originated from frequent itemset discovery [7], based on apriori-concepts, and was subsequently enhanced in various ways, such as achieving performance benefits through prefix representation of the dataset [8] or obtaining a more compact sequence representation by mining closed sequences [9]. Another algorithm that focused on reducing a number of redundant sequential patterns was proposed in [10] in the form of CCSpan, which adopts a snippet-growth paradigm and includes several pruning techniques to discover a compact set of closed contiguous sequential patterns.

The constraint-based approach cSPADE was introduced in [11], and has recently received a strong interest towards extending it along the declarative constraint programming paradigm. More specifically, several studies investigated how to generically build a knowledge base of constraints covering the sequences in a temporal dataset. For instance, [12] presented a satisfiability-based technique for enumerating all frequent sequences using cardinalities for the constraints retrieved. In [13], the authors introduced a general constraint programming approach with exists-embedding global constraint for sequences and its more general formulation, that steers away from explicit wildcards. This approach, however, requires significant computational resources. This was improved in [14] with a better prefix representation of sequence mining constraints, based on the prefix projection principle introduced in [8], and was later extended for GAP constraints in [15]. Similarly, [16] introduced an approach that speeds up the retrieval of constraints by precomputing the relations between items in a dataset to avoid reiterating over the sequences. These approaches also allow for fast retrieval of regular expressions.

A similar vein of research was pursued with `Warmr` [17], an inductive logic programming pattern discovery algorithm that relies on the `Datalog` formalization for expressing multi-dimensional patterns. It was elaborated further for sequences in [18]. The proposed work is a special purpose algorithm that mines for a subset of `Datalog` patterns.

2.2 Sequence Classification

While sequence mining and sequence classification share many common insights, the nature of the objective is different. Rather than extracting the full set of supported sequences or constraints, it is paramount that the feature set exhibits the following characteristics.

- **Informative:** features of sequential patterns should be supported in a database, but their usefulness towards classification, i.e. their discriminative power, also depends on other factors such as confidence and interestingness [19]. In general, there is a need for striking a balance in the feature set so that support values are in between extremely high and low values [20].
- **Concise:** the informative feature set is preferably small to aid comprehensibility (less features to grasp and relate to each other), while being comprehensive (capturing all information possible in the features) at the same time. This can be obtained by generating a small number of features that are informative enough to capture all information regarding the classes.

Many sequence classification algorithms have been developed [21], [22], [23], each focusing on a different approach ranging from extensions to sequential pattern mining algorithms, to statistical approaches that infer the explanatory power of subsequences. These algorithms can be either direct, in which case the features are extracted according to their usefulness for classification, or indirect, in which all features are firstly generated and subsequently provided as input to a classifier. In [21], the cSPADE algorithm was extended with an interestingness measure based on both the support and the window (cohesion) in which the items of the constraint occur. Similarly, BIDE was extended to BIDE-D(C) in [22] by incorporating information gain in order to provide a direct sequence classification approach. In [23], the sequence database is split up in smaller parts to be recreated by a sparse knowledge base that punishes for infrequent behavior by constructing a Bayesian network of posteriors that is able to reconstruct the sequence database. A similar approach is used in [24], where a strong emphasis is put on finding interesting sequences.

In contrast to simple support counting, the weight of information is used for finding interesting sequential patterns in weighted sequential pattern mining [25]. Based on this idea, a new method for mining time-interval weighted sequential (TiWS) patterns was proposed in [26]. Finally, [27] describes a method based on document-specific keyphrase extraction, which is able to capture semantic relationship between items.

In contrast to the previously mentioned techniques, iBCM draws from insights in constraint programming, but rather than constructing a complete constraint base that is able to regenerate the sequence database as a whole, iBCM employs highly diverse and informative behavioral patterns that incorporate cardinality, alteration, gaps, as well as negative information. By fixing the behavioral constraint template base, it is possible to devise a tailor-made and fast algorithm for retrieving them from large databases. The technique employs only binary constraint templates, however, other studies such as [21] have already revealed that for sequence classification, the length of sequential

Table 1: Example sequence database.

ID	Sequence	Label	ID	Sequence	Label
1	abbcaa	1	4	acbbcaacc	2
2	abbccaa	1	5	acbbcaa	2
3	abbaac	1	6	acbbcaa	2

patterns does not have to exceed 3, or even 2. However, the techniques can be extended to use higher-order behavioral constraints [28]. Finally, since sequence and especially text-based data, being time-based, are often prone to concept drift [29], [30], a window-based approach is devised as well.

3 The Framework of Behavioral Templates

In this section, we establish the preliminaries and give an overview of the behavioral constraint templates and their characteristics.

3.1 Sequences and Sequence Databases

Sequence classification involves the concepts of a sequence and a sequence database, as well as the labels needed to distinguish between the classes.

Definition 1. A sequence $\sigma = \langle \sigma_1, \sigma_2, ..., \sigma_n \rangle \in \Sigma^*$ is a list of items with length $|\sigma| = n$ out of the alphabet of items Σ, where * denotes Kleene star operator. We denote:

- $\sigma_a = \{\sigma \mid a \in \sigma, \sigma \in \Sigma\}$ the strings containing a,
- $occ(a, \sigma) = \{i \mid \sigma_i = a, i \in \mathbb{N}\}$ the ordered set of positions of item $a \in \Sigma$ in sequence σ,
- $min(occ(a, \sigma))$ the first occurrence of item a,
- $max(occ(a, \sigma))$ the last occurrence of item a, and
- $|occ(a, \sigma)|$ the number of occurrences of item a.

Sequences are typically bundled in sequence databases, which can be defined as follows.

Definition 2. A sequence database $\mathcal{SB}$ is a list of sequences with $L : \mathcal{SB} \rightarrow \mathbb{N}$ a labeling function assigning a class label to every sequence $\sigma \in \mathcal{SB}$. The number of sequences in the database is denoted $|\mathcal{SB}|$, and $l_{\mathcal{SB}} = |img(L(\mathcal{SB}))|$ is the number of different labels present in database $\mathcal{SB}$.

Consider the example sequence database in Table 1, with three items in the alphabet $\Sigma_{\mathcal{SB}} = \{a, b, c\}$, 6 sequences $|\mathcal{SB}| = 6$, and two labels $l_{\mathcal{SB}} = 2$.

3.2 The Declare Behavioral Constraint Template Base

iBCM is based on a set of behavioral constraint templates from the Declare language [31], which in turn is inspired by the formal verification patterns of Dwyer [32]. These patterns are widely used for identifying not only sequential, but overall behavioral characteristics of programs and processes. The Declare template base consists of a number of patterns for modeling flexible business processes, which are typically expressed in linear temporal logic (LTL), or regular expressions and finite state machines (FSMs). The template base is extensible, but the most widely-used entries, which are used in this paper, are listed in Table 2. The patterns contain both unary and binary constraints. The unary constraints focus either on the position (first/last), or the cardinality of items. The choice constraint can be considered as an *existence* constraint over multiple items, i.e., the occurrence of at least one of 2 items corresponds to *existence(a,b,1)*. The binary constraints exhibit a hierarchy [35]. There are unordered constraints (*responded/co-existence*), simple ordered (*precedence, response, succession*), alternating ordered, and chain ordered constraints. The presence of *chain precedence(a,b)* implies the presence of *alternate precedence(a,b)*, and *precedence(a,b)*, as well as *responded existence(b,a)*. This allows for expressing the presence of certain groups of items, their ordering, and also the repeating (alternation) and local (chain) behavior. Moreover, negative constraints are able to express that behavior does not occur, and, despite being proven especially useful in the context of classification, they are typically not generated by sequence classification techniques that only mine for positive patterns. A constraint can be defined formally as follows.

Definition 3. A sequence constraint $\pi = (A, t)$ is a tuple with A a set of items and t the type of constraint.

A binary constraint has an antecedent, implying the constraint, and a consequent. Both can exist out of a set of items, however, in the rest of the paper we will assume both to be singletons. The types of the constraints correspond with the templates that are defined in Table 2. For convenience, the constraints are written in an abbreviated fashion, e.g. $altPrec(a, b)$. They all correspond to a certain regular expression which can be converted into an FSM. We denote the corresponding regular expression as $\S(t)$.

Definition 4. A sequence σ supports a constraint π iff $\sigma \in \mathcal{L}(\mathcal{A}(\pi))$ where $\mathcal{L}$ denotes the language of the corresponding FSM. The support of the constraint in the database is $sup(\pi)_{\mathcal{SB}} = |\{\sigma | \sigma \in \mathcal{L}(\mathcal{A}(\pi)), \forall \sigma \in \mathcal{SB}\}|$.

E.g. in $\mathcal{SB} = \{aab, abb\}$, $\sigma_1 \in \mathcal{L}(\S(altPrec(a, b)))$, $\sigma_2 \notin \mathcal{L}(\S(altPrec(a, b)))$, and $sup(\pi)_{\mathcal{SB}} = 1$.

3.3 Motivation for Behavioral Constraint Templates

In this section, we detail the rationale behind the power of the behavioral constraint templates by explaining how each type of constraint template can improve expressiveness and conciseness of patterns when compared to sequential patterns used in classic sequence mining and classification algorithms. The latter techniques often employ ordered item sets or partial orders, thus essentially relying on a representation allowing for a wildcard-based language. E.g. a mined sequence such as *abba*, derived with any of the other techniques that are used in Section 5, would be labeled with a language expressed as [^ab]*a.*b.*b.*a[^ab]*. In the following sections, it is illustrated how the different behavioral constraint templates are more powerful than such a representation.

3.3.1 Negative (and choice) constraints

First, finding behavior that is not present cannot be expressed with partial orders. Indeed, the absence of an item, as well as the conditional presence of an item (e.g. all *responded existence*-based constraints which comprise all the ordered binary constraints) are impossible to include in a partial order. Nonetheless, algorithms tailored towards mining partial orders resort to finding sequences that approach the absence of items, by generating many sequences that

Table 2: An overview of Declare constraint templates with their corresponding LTL formula and regular expression.

Type	**Template**	**LTL Formula [33]**	**Regular Expression [34]**
Unary	Existence(A,n)	$\Diamond(A \wedge \bigcirc(existence(n-1, A)))$	.*(A.*){n}
	Absence(A,n)	$\neg existence(n, A)$	[^A]*(A?[^A]*){n-1}
	Exactly(A,n)	$existence(n, A) \wedge absence(n+1, A)$	[^A]*(A[^A]*){n}
	Init(A)	A	(A.*)?
	Last(A)	$\Box(A \implies \neg X \neg A)$	.*A
Unordered	Responded existence(A,B)	$\Diamond A \implies \Diamond B$	[^A]*((A.*B.*) |(B.*A.*))?
	Co-existence(A,B)	$\Diamond A \iff \Diamond B$	[^AB]*((A.*B.*) |(B.*A.*))?
Simple ordered	Response(A,B)	$\Box(A \implies \Diamond B)$	[^A]*(A.*B)*[^A]*
	Precedence(A,B)	$(\neg B\, U A) \vee \Box(\neg B)$	[^B]*(A.*B)*[^B]*
	Succession(A,B)	$response(A, B) \wedge precedence(A, B)$	[^AB]*(A.*B)*[^AB]*
Alternating ordered	Alternate response(A,B)	$\Box(A \implies \bigcirc(\neg A\, U\, B))$	[^A]*(A[^A]*B[^A]*)*
	Alternate precedence(A,B)	$precedence(A, B) \wedge \Box(B \implies \bigcirc(precedence(A, B))$	[^B]*(A[^B]*B[^B]*)*
	Alternate succession(A,B)	$altresponse(A, B) \wedge precedence(A, B)$	[^AB]*(A[^AB]*B[^AB]*)*
Chain ordered	Chain response(A,B)	$\Box(A \implies \bigcirc B)$	[^A]*(AB[^A]*)*
	Chain precedence(A,B)	$\Box(\bigcirc B \implies A)$	[^B]*(AB[^B]*)*
	Chain succession(A,B)	$\Box(A \iff \bigcirc B)$	[^AB]*(AB[^AB]*)*
Negative	Not co-existence(A,B)	$\neg(\Diamond A \wedge \Diamond B)$	[^AB]*((A[^B]*) |(B[^A]*))?
	Not succession(A,B)	$\Box(A \implies \neg(\Diamond B))$	[^A]*(A[^B]*)*
	Not chain succession(A,B)	$\Box(A \implies \neg(\bigcirc B))$	[^A]*(A+[^AB][^A]*)*A*
Choice	Choice(A,B)	$\Diamond A \vee \Diamond B$	.*[AB].*
	Exclusive choice(A,B)	$(\Diamond A \vee \Diamond B) \wedge \neg(\Diamond A \wedge \Diamond B)$	([^B]*A[^B]*) |.*[AB].*([^A]*B[^A]*)

are not containing the behavior that is absent. However, this only holds for sequences and not single items. The non-occurrence of an item in 100% of the sequences indicates the absence of that item, but this renders partial order-based approaches incapable of finding absence in only subsets of the sequences. Hence, for *not succession(a,b)*, sequences that do not contain b after an occurrence of a can be distinguished by a classifier by generating different partial orders that do not contain b after a. However, there would be plenty of partial orders needed to express this, while for iBCM *not succession* would be present as a single feature. For *absence(a,1)* it would be possible to distinguish sequences with and without a, but if there is no sequential information available that contains a, the non-occurrence of the item cannot be uncovered by partial orders.

For *not chain succession*, the same rationale as for the chain constraints can be followed. *Exclusive choice* and *not co-existence* follow a similar reasoning as for *absence*, while *choice(a,b)* can be approached in the same way as *existence(a,b)*, as explained below.

3.3.2 Position constraints

Last and *init* constraints are virtually impossible to detect for a simple partial order representation. Hence, mining a language such as a.* (init) or .*a (last) for one class and [^a].*a.* and .*a.*[^a] (last) for another class does not yield any discerning features when running partial order-based classifiers, while iBCM with behavioral constraints is capable of using only one constraint to distinguish both classes.

3.3.3 Existence constraints

Existence(a,n) constraints count the number of times n an item a occurs. In general, partial orders are also capable of representing this number by indicating how many consecutive occurrences of the item are present, e.g. a sequence aaa shows that a occurs three times. However, the interpretation of behavioral constraints is easier, since they are formally defined, easily explained in regular expressions and executable by automata. Furthermore, using the number of occurrences as a feature (*existence(a,3)*) would make classification easier than with simply occurrence *existence(a,1)*, and allows to reduce the number of features as well by excluding constraints through hierarchy reduction, e.g. excluding *existence(a,1)* as *existence(a,3)* is present. However, occurrence of aa is not removed as aaa holds, unless specifically implemented.

3.3.4 Unordered binary constraints

Responded existence is not used in iBCM, however, it forms the base for *precedence* and *response* constraints. [^a]*((a.*b.*)|(b.*a.*))? expresses that the presence of a requires the presence of b, without specifying its location in the string. This concept is hard to capture with partial orders, as multiple orders have to be incorporated to establish that the location is unimportant, hence requiring many different orders to approach the lack of importance of orders. Also the possible absence of a cannot be expressed, as this is negative information. *Co-existence* would require two partial orders, i.e. ab, and ba, or the presence of both activities, to capture the fact that although they occur together, the position of the items is not fixed. The constraint does not suffer from the possible absence of a or b.

3.3.5 Simple ordered constraints

Precedence and *response* form the base for partial orders. However, their simple setup is deceiving. While partial orders can capture sequences of items, simple order constraints are not capable of distinguishing very simple nuances such as *bbaabab* vs. *bbaababa*. Indeed, the difference between the extra occurrence of a at the end of the string, turns the sequence from one where a *precedence(a,b)* constraint holds into one in which a *response(a,b)* holds. It was tested by simulating both strings for different classes with different items interleaved and mining them with the techniques outlined in Section 5. Many were not capable of distinguishing the difference between the classes, unless many other patterns are generated as well. In a simulation, cSPADE is not able to distinguish both traces, and

PrefixSpan needs 41 patterns to summarize the two strings shown before. Moreover, the number of required sequences increases with the length of the traces. MiSeRe is unable to distinguish them as well, and the other algorithms SCIS and BIDE learn the sequences by hard and churn out the two simulated strings. Hence, they fail to capture the underlying cause, which is the different end, which can manifest itself in many different ways (it corresponds to the size of the language of both constraints). For *precedence*, a similar example can be constructed.

3.3.6 Alternating ordered constraints

For alternate constraints, the same nuances exist. Consider, for example, $\underline{abbabababb}$ and $\underline{abba}a\underline{bababb}$. The presence of one extra a makes *alternate response(a,b)* hold in the first, but not for the latter sequence where only *response(a,b)* holds. Again, the other algorithms are only capable of finding this distinction by learning the full sequences, rather than the concept of repetition. Furthermore, the different instantiations of an *alternate response* renders the partial order representation unable to distinguish shorter and longer occurrences of such repetitions, e.g. the difference between comparing *abbabababb* and *abbabababbabbabaababb* with *abbaabababb* would make partial order-based techniques generate even more features. The same line of thinking can be applied to *alternate precedence* as well, where in *abaabab alternate precedence(a,b)* holds and in *abaabb* only *precedence(a,b)*.

3.3.7 Chain ordered constraints

Typical partial order representations do not contain any notion of 'next' position, and allow for any item to fill the gap in between items in a sequence. If we use the same example as for alternate constraints, but generate one label with and one without gaps, none of the other algorithms is capable of distinguishing the strings generated.

4 iBCM: Algorithm Design and Implementation

This section outlines the algorithm for constructing the set of features based on the constraint templates discussed in Section 3. iBCM is an indirect sequence classification approach, i.e. the featurization and classification parts are separate. Afterwards, Section 5 will outline the classification performance of the constraint templates when used as binary input features (present/not present).

4.1 Featurizing Sequences with iBCM

Given that iBCM is an indirect sequence classification approach, in this section, we first outline the featurization part. For inferring potentially discriminative sequential patterns from a sequence database, iBCM applies a three step approach, as outlined in Algorithm 1. In a nutshell, the algorithm derives the sequential patterns per class of sequences by intelligently checking whether behavioral constraint templates hold or not.

4.1.1 Step 1: Retain frequent items

First, items that exceed the support threshold are withheld in set A (Alg. 1 line 3). Only these items are used for checking unary constraints, and subsequently checking binary constraints in pairs.

4.1.2 Step 2: Generate constraints

In the next step, all the sequences belonging to a particular class i with label l in the database are checked in the following manner (starting line 5, Alg. 1). First, if the length of the sequence $|\sigma|$ is smaller than the number of windows w, the sequence is not considered (line 6, Alg. 1). Then, constraints are mined per window (Alg. 2). This involves the computation of the number of items in a window and deriving the bounds (lines 3-4, Alg. 2). Finally, constraints are mined for every window.

Hereto, the (sub)sequence or window is traversed completely, and for every item in the sequence σ_i the positions are stored in $occ(\sigma_i, \sigma)$ (line 3, Alg. 3) to keep track of the position of the *occ*urrences of an item σ_i in σ. The number of occurrences is denoted $|occ(\sigma_i, \sigma)|$. This allows for easy verification of the constraints, as the position in the string is used to quickly check the number of occurrences for unary constraints, and the position relative to the other items for binary constraints. For every item $a \in A$, $|occ(a, \sigma)|$ is used for determining the cardinality constraints, i.e. *absence/exactly/existence* (lines 5-7, Alg. 3) for *existence/absence/exactly* constraints. It is also checked whether it occurred as the first or last item in the sequence (lines 8-9, Alg. 3) for checking $init$ and $last$ constraints. Next, a is paired with every other $b \in A \setminus a$ to determine the type of behavioral constraint pattern. First, co-existence of a and b is checked. Next, if a occurs before b, the precedence hierarchy is reviewed (lines 12-21, Alg. 3). For every next occurrence of b, it is checked whether there was another a preceding it for *alternate precedence*. In the meantime for every occurrence, the exact position is checked for *chain precedence*. If all occurrences of b are in the right position for the constraint to hold, it is added to the constraint set. Both checks stop when there is no further evidence, hence constraints can be verified efficiently with minimal look ups.

If b occurs after a, the response hierarchy is scrutinized (lines 22-34, Alg. 3). Similar to *alternate precedence*, every occurrence of a is checked for a subsequent b before the next occurrence. If every next occurrence of a is b, *chain response* is stored. After every pairwise check, the respective *succession* constraints are added if both *(alternate/chain) response* and *precedence* are present in the sequence. When b is not present in the sequence, there is evidence for *exclusive choice*. Afterwards, in line 35 of Algorithm 3, constraints are merged if they can be combined into a constraint higher up the hierarchy (i.e. the simultaneous presence of *response* and *precedence* forms *succession*) to reduce the size of the number of features. Since the algorithm is focused on only retaining constraints when both items are present, all constraints are mined with 100% confidence. Techniques that take confidence for Declare constraints into account while avoiding discovering vacuous constraints exist as well [35], [36], [37]

4.1.3 Step 3: Retain frequent constraints

Finally, for every constraint it is checked whether it satisfies the minimum support level for the different labels in the sequence database in line 9 of Algorithm 1. This allows for the precise measuring of sequential behavior, as some sequences might support both *response* and *precedence*, and others might not. Distinguishing between the slight nuance between these two constraints allows for a better capturing of the distinctive behavior present for different labels.

The binary constraints can be derived very efficiently by boolean and string operations, as it is clear from Algorithm 3, which is inspired by [35] and [36]. However, for classification purposes the sequences need to be labeled right away. In [36], Büchi automata based on the LTL formulas are used to check constraints for each frequent pair. Doing this on a sequence level is computationally expensive, as it would require running each string many times. In [35], a knowledge base of occurrence and precedence relations is built and the support for constraints is calculated, which avoids single string run-throughs, but does not label strings individually for the presence of the constraints. iBCM rather analyzes the strings in one run, hence avoiding replaying strings over separate automata for each constraint, and obtaining the information regarding the features immediately.

Finally, on line 11 of Algorithm 1, the hierarchy reduction is applied, where all the constraints are subsumed by constraints down the hierarchy, e.g. *alternate response(a,b)* removes *response(a,b)*, as well as *responded existence(a,b)*, and *existence(a,3)* removes *existence(a,2)*, and *existence(a,1)*.

Algorithm 1 Mining constraint features per class i

```
1:  procedure RETRIEVE_CONSTRAINTS($\mathcal{SB}, minsup, w$)      ▷ Input: Data,
    minimum support, and number of windows
2:      for $l \in [1, l_{\mathcal{SB}}]$ do
3:          $A \leftarrow \{a \mid a \in \Sigma, |\sigma_a| \geq minsup\}$      ▷ STEP 1
4:          $C_l \leftarrow \emptyset$      ▷ $C_l$ a list with the constraints supporting label $l$
5:          for $\sigma \in \mathcal{SB} \wedge L(\sigma) = l$ do      ▷ STEP 2
6:              if $|\sigma| \geq w$ then
7:                  $C_l \leftarrow C_l \cup mineConstraints(\sigma, A, w)$
8:          for $c \in C_l$ do      ▷ STEP 3
9:              if $|\{c|c \in C_l\}| \geq |C_l| \times minsup$ then
10:                 $C_{\mathcal{SB},l} \leftarrow C_{\mathcal{SB},l} \cup c$
11:     applyHierarchyReduction
12:     return $C_{\mathcal{SB}}$
```

Algorithm 2 Mining constraints in a string per window

```
1:  procedure MINECONSTRAINTS($\sigma, A, w$)
2:      $C \leftarrow \emptyset$      ▷ C is a set of constraints
3:      $winSize = \lfloor \frac{|\sigma|}{w} \rfloor$
4:      $bounds \leftarrow \{ub \mid ub \mod w = 0, ub \leq |\sigma|\} \cup |\sigma|$
5:      $sort(bounds)$
6:      $i \leftarrow 0$
7:      for $b \in bounds$ do
8:          $\sigma_{i,b} \leftarrow \{\sigma_s \mid \sigma_s \in \sigma, s \geq winSize \times i + 1, s \leq b\}$
9:          $C \leftarrow C \cup mineConstraintsInWindow(\sigma_{i,b}, A)$
10:         $i{+}{+}$
11:     return $C$
```

Algorithm 3 Mining behavioral constraint templates

```
1:  procedure MINECONSTRAINTSINWINDOW($\sigma, A$)
2:      $C \leftarrow \emptyset$      ▷ $C$ is a set of constraints
3:      for $\sigma_i \in \sigma$ do $occ(\sigma_i, \sigma) \leftarrow i$
4:      for $a \in A \cap \Sigma_\sigma$ do      ▷ $\Sigma_\sigma$ is the alphabet of the sequence
5:          if $|occ(a,\sigma)| = 0$ then $C \leftarrow C \cup absence(a,1)$ ▷ Unary constraints
6:          else if $|occ(a,\sigma)| > 2$ then $C \leftarrow C \cup existence(a,3)$
7:          else $C \leftarrow C \cup exactly(a, |occ(a,\sigma)|)$
8:          if $1 \in occ(a,\sigma)$ then $C \leftarrow C \cup init(a)$
9:          if $|\sigma| \in occ(a,\sigma)$ then $C \leftarrow C \cup last(a)$
10:         for $b \in \{A \cap \Sigma_\sigma\} \setminus \{a\}$ do      ▷ Binary constraints
11:             $C \leftarrow C \cup CoExist(a,b)$
12:             if $min(occ(a,\sigma)) < min(occ(b,\sigma))$ then
13:                 $C \leftarrow C \cup prec(a,b)$
14:                 $i \leftarrow min(occ(b,\sigma))$
15:                 $chain \leftarrow (i-1) \in occ(a,\sigma),\ continue \leftarrow \top$
16:                 while $\exists n \in occ(b,\sigma),\ n > i \wedge continue$ do
17:                     if $\exists p \in occ(a,\sigma), i < p < n$ then $i \leftarrow n$
18:                         if $\neg chain \vee (n-1) \notin occ(a,\sigma)$ then $chain \leftarrow \neg$
19:                     else $continue \leftarrow \neg$
20:                 if $continue \wedge |occ(b,\sigma)| > 1$ then $C \leftarrow C \cup altPrec(a,b)$
21:                 if $chain$ then $C \leftarrow C \cup chainPrec(a,b)$
22:             if $max(occ(a,\sigma)) < max(occ(b,\sigma))$ then
23:                 $C \leftarrow C \cup resp(a,b)$
24:                 if $max(occ(a,\sigma)) < min(occ(b,\sigma))$ then
25:                     $C \leftarrow C \cup notSuc(a,b)$
26:                 $i \leftarrow min(occ(a,\sigma))$
27:                 $chain \leftarrow (i+1) \in occ(b,\sigma),\ continue \leftarrow \top$
28:                 while $\exists n \in occ(a,\sigma),\ n > i \wedge continue$ do
29:                     if $\exists p \in occ(b,\sigma), i < p < n$ then
30:                         $i \leftarrow n$
31:                         if $\neg chain \vee (n+1) \notin occ(b,\sigma)$ then $chain \leftarrow \neg$
32:                     else $continue \leftarrow \neg$
33:                 if $continue \wedge |occ(a,\sigma)| > 1$ then $C \leftarrow C \cup altResp(a,b)$
34:                 if $chain$ then $C \leftarrow C \cup chainResp(a,b)$
35:             add succession if (alternate/chain) response and precedence
36:             if $b \notin \Sigma_\sigma \wedge b \in A$ then $C \leftarrow C \cup exclChoi(a,b)$
37:     return $C$
```

The window-based nature of the approach allows to find constraints in different parts of the sequence, hence being able to capture underlying changes in the system generating the items. By varying the window parameter w, the algorithm can be made more sensitive to changes in the dataset, being capable of dealing with both abrupt shocks, or more gradual changes [38]. Consider, for example, *ababaabbab*. In the first 5 steps, *alternate precedence(a,b)* holds, while in the latter 5 steps, the sequence has shifted towards a *alternate response(a,b)*-based pattern. Overall, neither of these constraints hold. iBCM is capable of capturing such nuances if they are informative for distinguishing classes.

Once all constraints are obtained per class label, i.e., all $C_{SB,l}$ are found, they can be used for classification. First, constraints present in all classes are removed from all $C_{SB,l}$ as they are not discriminative. Next, the presence of every constraints in $C = \bigcup_l C_{\mathcal{SB},l}$ can be used as a binary feature for unseen sequences for classification.

4.2 Considerations on the Constraint Template Base

Not all Declare constraint templates are suitable to be considered for obtaining features from single sequences. First of all, constraints might suffer from being vacuously satisfied, i.e. they are satisfied because no counterevidence is provided. Hence, only binary pairs that are both present in a sequence are considered. This automatically satisfies the *choice* constraint, as well as *responded existence* and *co-existence*. Secondly, in a single sequence, *absence, exactly*, and *existence* are not distinguishable. It is opted not to generate all of them, but rather to stick with a layered approach of *absence* for no occurrences, *exactly* for 1 to 2 occurrences, and *existence* for more than 3 occurrences. It would be possible to check them separately, and merge them afterwards, however, experiments showed that this does not have an impact on the results. Finally, *exclusive choice* and *not chain succession* both mine for negative behavior that reflects everything that is not present in the sequences. While absence does the same, the magnitude of the number

Table 3: The behavioral constraints present in the sequence database of Table 1. The constraints that are supported at 100% are left out for 50%.

Support	Label	Supported constraint templates
100%	1	init(a), existence(a,3), exactly(b,2), response(b,a), precedence(a,c), **succession(b,c)**, **not succession(c,b)**, precedence(a,b)
	2	init(a), existence(a,3), exactly(b,2), response(b,a), precedence(a,c), precedence(c,b), response(b,c), **precedence(a,b)**
50%	1	exactly(c,1), last(a), response(c,a), **alternate precedence(a,c)**, **alternate precedence(b,c)**,
	2	last(a), **exactly(c,2)**, response(c,a)

of non-existing sequence pairs is vastly larger. Although mining for negative information is one distinctive feature of the proposed approach, the gain in accuracy performance does not outweigh the burden in terms of the number of extra constraints generated. Hence, these constraints are not included in the final constraint set. *Not succession* is the only negative constraint used.

4.3 Comparison with Other Sequence Constraint Representations

The iBCM approach is not intended to be able to reproduce the database, but rather to capture the most discerning sequence-based features. Consider for example the database in Table 1. Table 3 lists the constraints present for both labels. For label 1, *a* does not always precede *b*. Also, for label 2, *c* occurs before *b*. This can be discerned by only three constraints per support level, which are marked in bold. Hence, with only three features, it is possible to classify the traces correctly. Lowering the support threshold results in more constraints being different, although the number of constraints does not have to drastically increase, as for example *response(a,b)* will eventually be replaced by *alternate response(a,b)* because of the contraints' hierarchy. It is harder to achieve the same results with typical sequence-based constraints in algorithms such as cSPADE, as non-local information that is present in some constraints, e.g. *succession*, requires either longer or more sequences to approach the behavior that will converge towards the language of the regular expression. This is illustrated below.

4.4 Scalability

The computational tractability of the technique relies heavily on two components. First of all, the length of the sequence is an important factor as every sequence is traversed completely. Hence, the performance is bound in the extreme by the length of the longest sequence. Secondly, the minimum support determines the number of activities, hence the number of pairs and constraint templates that need to be checked. In the worst case, all pairs have to be checked for all binary templates. Most constraints can be checked by simple lookups, but in case the templates in the upper part of the hierarchy are checked, the complexity in the worst case is the length of the string for checking alternating and chain behavior. This results in O($|A|^2 \times max(|\sigma|)$). However, as it is clear from experimental evaluation, iBCM can achieve good results at high minimum support levels, reducing $|A|$ drastically.

5 Experimental Evaluation

In this section, iBCM is evaluated against eight state-of-the-art sequence classification algorithms using six widely-used datasets.

5.1 Setup

Below, we provide an overview of the datasets, the benchmark sequence classification techniques, and the implementation.

5.1.1 Data

The datasets are summarized in Table 4. They vary significantly in terms of the number of sequences, distinct items and classes, and also in terms of the average and maximum length of the sequences. For more details about the datasets, we refer to [23] and [21].

Table 4: Characteristics of the datasets used for comparative evaluation.

	$\lvert\mathcal{SB}\rvert$	$\lvert\mathbf{\Sigma}_{\mathcal{SB}}\rvert$	$l_{\mathcal{SB}}$	$\mathbf{avg}(\lvert\sigma\rvert)$	$\max(\lvert\sigma\rvert)$
aslbu	424	250	7	13.05	54
auslan2	200	16	10	5.53	18
context	240	94	5	88.39	246
pioneer	160	178	3	40.14	100
reuters	4,976	27,884	5	139.96	6,779
Unix	5,472	1,697	4	32.34	1,400

5.1.2 Benchmark Sequence Classification Techniques

iBCM is benchmarked against eight state-of-the-art techniques, namely cSPADE [11], GoKrimp [39], Interesting Sequence Miner (ISM) [23], Sequence Classification based on Interesting Sequences (SCIP) [21], BI-directional Extension (BIDE) [9], Prefix-projected Sequential pattern mining (PrefixSpan) [8], and Mining Sequential Classification Rules (MiSeRe) [24]. Furthermore, we test Long Short-Term Memory networks (LSTMs) [40] for their prevalence and recent popularity for modelling sequences. For the indirect approaches, i.e. the algorithms generating sequences without considering the label, cSPADE, PrefixSpan, BIDE, iBCM, and SCIP, the support levels were set at 0.1-1.0 by 0.1 intervals. SCIP was used for a minimum interestingness level of 0.05 and a maximum sequence length of 2 (this length was devised by the authors in [21], and a longer length increased computation time and did not return better results)[1]. The direct approaches, i.e. algorithms that generate sequences according to their discriminative power for the labels, have different parameters. For MiSeRe, 1, 2, 5, and 10 second run times were considered. GoKrimp was used both in supervised and unsupervised fashion for generating the sequences. ISM was applied with a maximum number of iterations of 200, and a maximum number of optimization steps of 10,000. No notable differences were reported when using different settings. LSTMs were trained for both 3 and

[1]The standard implementation performs cross-validation on all sequences. This was reported in the first iBCM paper [6]. This results in more constraints, as they all have different levels of interestingness. In this paper, this cross-validation was dismissed and all sequences were mined in 1 set to be more comparable with the other techniques. This resulted in less constraints, but very poor performance in accuracy. For more details on SCIP, we refer to [6] and the original paper of [21].

50 epochs, to illustrate the difference in accuracy achieved by longer learning times. Finally, iBCM was tested for a number of window settings w, where $w = 5$ performed best in terms of accuracy and the number of constraints generated. In the results, both $w = 1$ and $w = 5$ are included, and for the latter only the results of classifying with random forests were included as there was no significant difference in terms of accuracy when using the other classifiers.

5.1.3 Implementation

All techniques first generate interesting sequential patterns. Next, a predictive model is built by using the obtained sequential patterns as binary features for the sequences. More specifically, for all the techniques, sequential patterns were first extracted to be used as features for the sequences, which were labelled according to whether the sequential pattern (feature) is present in the respective sequence or not. All features that were common for all the different classes were removed, as they do not have any discriminative power. Three classifiers were considered, namely naive Bayes (NB), support vector machines (SVM) and random forests (RF), for which the Weka[2] Java implementation was used.

All techniques have been implemented in Java by the respective authors, while cSPADE, GoKrimp, BIDE and PrefixSpan were retrieved from the SPMF library [41]. LSTMs were implemented using the Keras Python library[3]. The implementation of the benchmark can be found online[4]. All runs were executed using a Java 8 Virtual Machine on an Intel Xeon E3-1230 (v5) CPU with 32GB DDR4 memory. A 10-fold cross-validation was applied to all the experiments. All support-based algorithms were prevented from running longer than 30 minutes (and are indicated as NA in case they ran out of time) to generate the sequential patterns to be used for classification. All timings were recorded after the data was loaded and the sequence database established. Every algorithm (except LSTMs) was allowed only one thread to execute, while the 8 thread results of iBCM are included (with the tag MT). All ExecutorService objects were retrieved from the code and set to use only 1 thread, however, cSPADE nevertheless used a full multithreaded approach that could not be prevented. BIDE also launched more than 1 thread for a negligible part of the execution.

5.2 Results

The results in terms of accuracy, the number of generated constraints, and the best performing classifier can be found in Table 6. The full results can be found in Appendix A. The results for the direct approaches are reported in Table 5, as they do not use support as a parameter. The best results for every algorithm for every dataset are indicated in bold, and the best performing algorithm in terms of accuracy for the lowest number of constraints over all other algorithms (over the two tables) is indicated in red.

5.2.1 Accuracy and Number of Constraints Generated

Overall, iBCM is capable of achieving a high accuracy for all the datasets, outperforming other sequence mining algorithms. Most notably, for datasets *aslbu*, *auslan2*, and *Unix*, iBCM and its window-based variant perform significantly better in terms of accuracy and the amount of constraints generated. iBCM with a window of 1 is outperformed on a few occasions by PrefixSpan (e.g. for *pioneer* PrefixSpan achieves high accuracy sooner with less constraints). In general, iBCM retrieves a slightly larger constraint set than BIDE, SCIP and cSPADE, and typically more constraints than PrefixSpan for a higher support level, but less for a lower support level. For a higher window size, the constraint set is smaller, and the accuracy is the highest. iBCM and SCIP are the only techniques that are capable of finding features, and that do not run out of time or memory in case the support values are low. For the *context* dataset, they are the only algorithms that retrieve constraints timely. The others also fail to produce any constraints for higher constraint values, e.g. for *aslbu* and *Unix*.

While the direct approaches typically achieve high accuracy, they generate significantly more constraints due to their setup, i.e. mining as many interesting constraints in as little time possible. Especially ISM seems to provide generally strong results for which is does need a large set of features and performs similar, or outperforms GoKrimp and MiSeRe. LSTMs do not seem to add competitive results, although the number of epochs drastically improves the results, albeit at a significant runtime cost. In general, the level of accuracy of the direct approaches is still not as high as the one produced by iBCM's constraints for *aslbu*, *auslan2*, and *Unix*.

The other techniques provide sequence constraints that are less suitable for classification, as the results for the different classifiers tend to be less stable. While not reflected in the table, especially for PrefixSpan and cSPADE, the classifiers report different accuracy values (note that the full results can also be found on the iBCM website[4]). This is especially noticeable for *aslbu*, *auslan2*, and *Unix*. On the contrary, iBCM does not have this issue, as the values of accuracy produced by different classifiers are indistinguishable.

Especially for *reuters*, text-based data, and *pioneer*, sensor-based data, the difference between iBCM with $w = 1$ and $w = 5$ is significant. This suggests that for data sources that are prone to concept drift [38] iBCM is indeed capable of performing adequately. Still, it does not suffer from a reduction in accuracy when no drift is present, as the window-based approach always performs equally well, or outperforms the single window approach.

5.2.2 Execution time

The execution times are reported in $log(time)$ milliseconds and can be found in Figure 1, showing that many techniques produce their results very quickly. iBCM typically outperforms the other algorithms, and multi-threaded implementations of the algorithms as well outperform the single threaded for the larger database ($w = 5$ is also performed in multi-threaded fashion). Otherwise, thread management slightly hinders fast execution. iBCM scales well with the size of the database, and exponentially when the alphabet grows in size which can be seen from the results of *reuters*. The algorithm always goes through the full strings, but can quickly decide on what type of constraints are present. Only if constraints higher in the hierarchy are present (alternating or chain constraints), iBCM has to perform multiple item

[2]http://www.cs.waikato.ac.nz/ml/weka/
[3]https://keras.io/
[4]https://feb.kuleuven.be/public/u0092789/

Table 5: Accuracy results of the best classifier for MiSeRe, ISM, LSTMs, and GoKrimp. All durations are log(time).

dataset	MiSeRe 1s (#)	MiSeRe 2s (#)	MiSeRe 5s (#)	MiSeRe 10s (#)	ISM (#)	LSTM 3 ep. (ms)	LSTM 50 ep. (ms)	GoKrimp unsup. (#-ms)	GoKrimp sup. (#-ms)
aslbu	0.55 (127)	0.55 (129)	0.552 (129)	0.55 (129)	**0.623 (2274)**	0.376 (3.298)	0.576 (4.049)	0.482 (10 - 2.474)	0.478 (10 - 2.417)
auslan2	**0.32 (252)**	0.32 (252)	0.32 (252)	0.32 (252)	0.26 (2401)	0.15 (3.223)	0.3 (3.571)	0.255 (3 - 1.23)	0.256 (3 - 1.447)
context	0.936 (590)	0.928 (1151)	0.935 (2653)	**0.94 (4860)**	0.888 (3568)	0.146 (3.673)	0.417 (4.685)	0.89 (38 - 4.128)	0.89 (38 - 4.143)
pioneer	0.956 (82)	0.91 (123)	0.852 (210)	0.823 (268)	**1 (2342)**	0.594 (3.327)	0.938 (3.336)	0.937 (18 - 2.371)	0.933 (18 - 2.32)
reuters	0.93 (2783)	0.93 (2783)	0.93 (2792)	0.93 (2800)	**0.971 (3445)**	0.594 (3.3)	0.765 (7.754)	0.63 (150 - 6.629)	0.63 (150 - 6.625)
Unix	0.865 (299)	0.54 (310)	0.716 (661)	0.7 (701)	0.91 (2842)	0.9 (5.347)	**0.924 (6.571)**	0.678 (78 - 5.308)	0.678 (78 - 5.337)

Table 6: An overview of the accuracy produced by the best classifier (in brackets) and the number of sequence constraints generated by the different algorithms for the 6 datasets. The highest accuracy with the lowest number of constraints for every classifier for every dataset is indicated in bold, and the best performing classifier per dataset in red.

		Technique											
dataset	**support**	**SCIS**		**iBCM (w=5)**		**BIDE**		**PrefixSpan**		**cSPADE**		**iBCM**	
		acc	**#con**	**acc**	**#con**	**acc**	**#con**	**acc**	**#con**	**acc**	**#con**	**acc**	**#con**
aslbu	**0.2**	**0.663 (NB)**	**59**	0.952 (RF)	330	**0.788 (RF)**	**134**	**0.937 (RF)**	**1930**	**0.812 (NB)**	**245**	0.967 (NB)	455
	0.4	0.483 (NB)	8	**0.994 (NB)**	**56**	0.695 (NB)	26	0.779 (NB)	97	0.700 (SVM)	45	**0.994 (RF)**	**86**
	0.6	NA	NA	0.569 (NB	23	0.428 (RF)	6	0.474 (NB)	12	0.398 (RF)	12	0.526 (NB)	17
	0.8	NA	NA	NA	NA	0.416 (SVM)	1	0.437 (RF)	3	0.405 (NB)	3	0.394 (NB)	1
	1	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
auslan2	**0.2**	0.2 (NB)	29	0.580 (SVM)	163	**0.545 (NB)**	**47**	NA	NA	0.380 (NB)	78	**0.698 (RF)**	**130**
	0.4	**0.2 (NB)**	**21**	0.675 (SVM)	90	0.460 (RF)	15	**0.420 (RF)**	**831**	**0.420 (NB)**	**58**	0.602 (SVM)	89
	0.6	NA	NA	**0.706 (NB)**	**52**	0.335 (NB)	5	0.390 (RF)	128	0.255 (NB)	30	0.5493 (SVM)	39
	0.8	NA	NA	0.652 (RF)	21	0.275 (RF)	3	0.405 (RF)	48	0.205 (RF)	14	0.358 (NB)	15
	1	NA	NA	0.383 (SVM)	10	0.095 (NB)	2	0.120 (RF)	12	0.105 (RF)	7	0.26 (NB)	9
context	**0.2**	0.903 (NB)	762	**0.992 (RF)**	**4339**	NA	NA	NA	NA	NA	NA	0.971 (RF)	4603
	0.4	0.900 (SVM)	562	0.991 (NB)	2202	NA	NA	NA	NA	NA	NA	0.974 (RF)	2932
	0.6	0.892 (RF)	349	0.99 (SVM)	1268	NA	NA	NA	NA	NA	NA	0.98 (SVM)	2151
	0.8	**0.966 (NB)**	**106**	0.99 (SVM)	663	NA	NA	NA	NA	NA	NA	0.988 (RF)	1534
	1	0.412 (NB)	6	0.965 (NB)	121	NA	NA	NA	NA	NA	NA	**0.996 (SVM)**	**632**
pioneer	**0.2**	0.994 (NB)	374	**0.993 (NB)**	**773**	0.975 (RF)	3434	0.981 (RF)	9804	0.988 (RF)	1645	0.973 (SVM)	3616
	0.4	0.95 (NB)	103	0.856 (RF)	357	0.981 (RF)	228	1.000 (RF)	602	0.994 (SVM)	218	1.000 (NB)	436
	0.6	0.838 (NB)	42	0.893 (NB)	78	0.994 (RF)	64	1.000 (SVM)	161	0.988 (RF)	55	0.831 (NB)	87
	0.8	1.000 (NB)	6	0.899 (NB)	5	0.762 (RF)	2	1.000 (NB)	9	0.787 (NB)	6	0.769 (NB)	8
	1	**1.000 (NB)**	**6**	NA	NA	**1.000 (NB)**	**2**	**1.000 (NB)**	**6**	**1.000 (NB)**	**6**	**1.000 (NB)**	**5**
reuters	**0.2**	**0.887 (NB)**	**75**	0.976 (RF)	1443	NA	NA	NA	NA	**0.960 (RF)**	**621**	1.000 (NB)	1213
	0.4	0.837 (NB)	23	0.988 (SVM)	412	**0.911 (RF)**	**109**	**0.994 (RF)**	**963**	0.915 (RF)	89	**1.000 (RF)**	**186**
	0.6	NA	NA	**0.998 (RF)**	**196**	0.852 (RF)	21	0.982 (RF)	59	0.861 (RF)	21	0.958(SVM)	41
	0.8	NA	NA	0.990 (SVM)	67	0.548 (RF)	2	0.920 (SVM)	6	0.564 (NB)	2	0.250 (RF)	1
	1	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Unix	**0.2**	0.817 (NB)	54	**1.000 (RF)**	**389**	**0.902 (RF)**	**59**	NA	NA	**0.902 (RF)**	**59**	**1.000 (NB)**	**118**
	0.4	**0.826 (NB)**	**9**	0.998 (NB)	113	0.851 (RF)	11	**0.945 (RF)**	**32**	0.835 (SVM)	11	0.996 (RF)	30
	0.6	0.760 (RF)	3	0.981 (NB)	24	0.771 (NB)	3	0.822 (SVM)	4	0.763 (SVM)	3	0.9184 (NB)	18
	0.8	NA	NA	0.573 (NB)	12	NA	NA	NA	NA	NA	NA	0.541 (NB)	8
	1	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

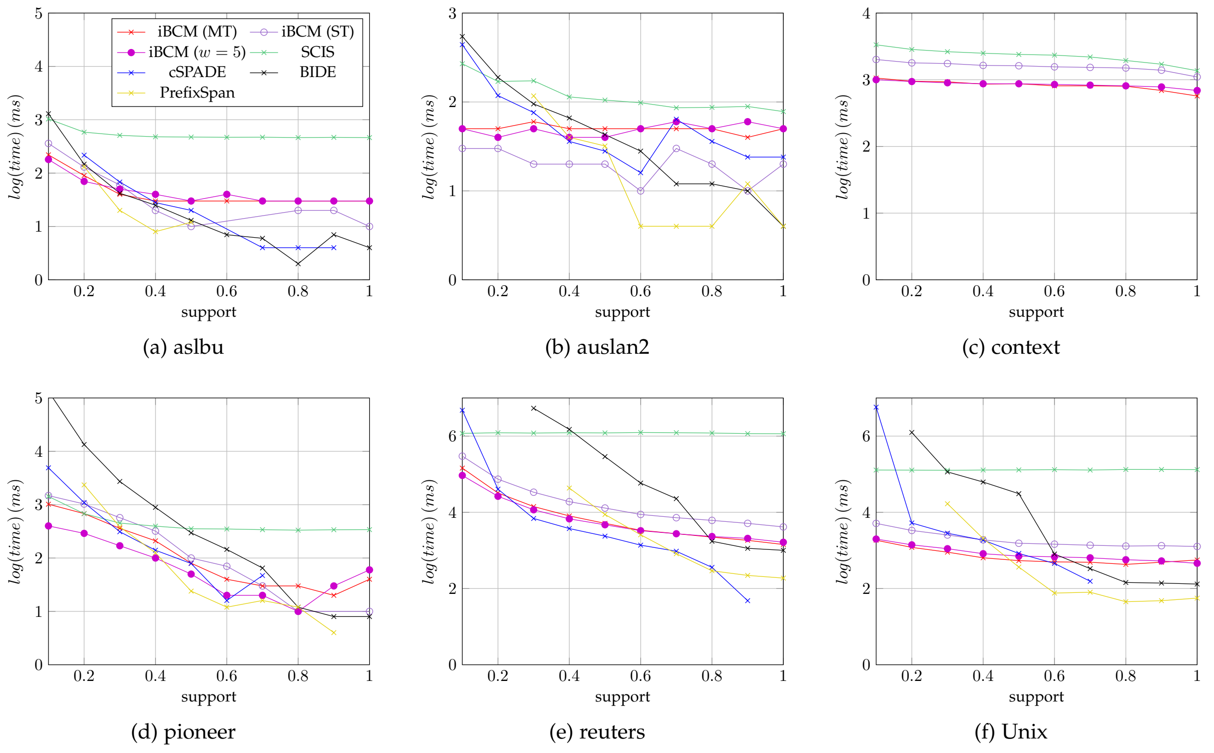


Figure 1: Overview of the performance of the different algorithms.

position checks (see Algorithm 3). Since those constraints are not as prevalent, the execution is typically unaffected by support, but this makes iBCM slower for higher support values, and fast for lower support values. However, the difference is negligible, and for larger datasets iBCM clearly outperforms the other algorithms. Overall, BIDE and PrefixSpan perform competitively for higher support values.

5.3 Information Gain

To get a better understanding of the contribution of each behavioral constraint template to the classification, we report the average information gain [42] for the different levels of support. Note that, in this case, the window parameter of iBCM was set to the default of $w = 1$. The overview can be found in Figure 2. The results for alternating ordered constraints for *Unix*/*auslan2*/*aslbu* and the results for position constraints for *context*/*pioneer*/*reuters* were omitted because the scores were always below 0.1. The information gain can be used to explain the results obtained from the classification:

- Simple ordered constraints are somewhat informative in most datasets, especially for *auslan2*, *pioneer*, and *context*.
- It appears that, in general, alternate constraints are not generating many insights, except for the *context* dataset.
- Chain constraints, especially the response and succession versions, perform well for *auslan2* and peak for *context*.
- The last constraint is informative in the *Unix* dataset.
- The existence(1) constraint is the most informative for most of the datasets, but existence(3) proves informative for *context*.
- The absence and exactly constraints perform well overall. Absence even scores consistently over the whole support spectrum.
- Co-existence and not succession are both informative for *auslan2*, and especially for the *context* and *pioneer* datasets.

Taking into account these insights, the performance in Table 6 can be further explained: iBCM is better capable of gaining insights in *aslbu* (lower support), *auslan2*, and *Unix*. The first observation can be linked to the presence of simple ordered, as well as existence, but probably absence constraints being generated. The second observation regarding *auslan2*, can be linked to the overall presence of a more varied set of constraints which is still smaller than the second best contender (PrefixSpan). Many negative constraints are generated as well (*absence/not succession*). Finally, the use of *absence* and *exactly* constraints make up for the strongest informative features for the *Unix* dataset.

Overall, it can be concluded that the introduction of negative constraints indeed increases the accuracy in places where other sequence classification algorithms fall short in terms of expressiveness. The results also show that there might be little need for sequential information to be generated, even in sequential data, as the alternate constraints are not informative, and the *exactly/existence/absence* constraints perform well in terms of information gain. This also leaves the opportunity to fine-tune the types of constraints used in iBCM (i.e. mainly the negative information is informative).

Note that *co-existence* is not part of the final constraint set, however, its performance overlaps with *existence/exactly* constraints as was indicated in Section 4.2.

5.4 Conclusion

In general, we can conclude that iBCM outperforms the other classifiers either by providing better accuracy, fewer constraints, faster runtime, or all three at the same time. Especially text-based datasets can be mined more accurately and faster for higher support values and with fewer constraints, as becomes apparent from the *aslbu*, *auslan2*, *reuters*, and *Unix* datasets. Clearly, the inclusion of negative constraints increases the accuracy, and the window-based approach pays off in many occasions as well.

6 Conclusion and Future Work

This paper presented iBCM, a novel powerful sequence classification algorithm. It extends our initial approach in several ways, but most importantly by making the technique window-based so as to accommodate for concept drift. iBCM is driven by a rich set of behavioral constraint templates, which are checked with respect to the sequences in a database. In an extensive comparative experimental evaluation, it is shown that iBCM is able to improve classification accuracy and runtime because of its increased expressiveness and conciseness. Furthermore, it can also be applied towards descriptively interpreting the nature of the sequential patterns present in a sequence database, offering insights into what types of interplay between items are present in the data.

For future work, we foresee several directions. First of all, a further in-depth comparison of the types of constraints that contribute most to the classification task for different types of datasets could lead to devising a direct sequence classification technique. Next, data-aware versions of the constraint templates could be leveraged for more complex sequential databases. Finally, other important areas of future investigation exist with respect to including non-sequential information [43] to bridge the gap with `Datalog` [17], and the target-branched version of Declare [28], which would allow for constraint templates with a consequent being a set rather than a singleton.

References

[1] J. T. Wang, S. Rozen, B. A. Shapiro, D. E. Shasha, Z. Wang, and M. Yin, “New techniques for DNA sequence classification,” *Journal of Computational Biology*, vol. 6, no. 2, pp. 209–218, 1999.

[2] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. J. C. H. Watkins, “Text classification using string kernels,” *Journal of Machine Learning Research*, vol. 2, pp. 419–444, 2002.

[3] J. Lee, J. Han, X. Li, and H. Cheng, “Mining discriminative patterns for classifying trajectories on road networks,” *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 5, pp. 713–726, 2011.

[4] F. Eichinger, D. D. Nauck, and F. Klawonn, “Sequence mining for customer behaviour predictions in telecommunications,” in *Proceedings of the Workshop on Practical Data Mining at ECML/PKDD*, 2006, pp. 3–10.

[5] M. Jaber, P. T. Wood, P. Papapetrou, and A. González-Marcos, “A multi-granularity pattern-based sequence classification framework for educational data,” in *DSAA*. IEEE, 2016, pp. 370–378.

[6] J. De Smedt, G. Deeva, and J. De Weerdt, “Behavioral constraint template-based sequence classification,” in *ECML/PKDD*, ser. Lecture Notes in Computer Science. Springer, 2017.

[7] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in *SIGMOD Conference*. ACM Press, 1993, pp. 207–216.

[8] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, "Prefixspan: Mining sequential patterns by prefix-projected growth," in *ICDE*. IEEE Computer Society, 2001, pp. 215–224.

[9] J. Wang and J. Han, "BIDE: efficient mining of frequent closed sequences," in *ICDE*. IEEE Computer Society, 2004, pp. 79–90.

[10] J. Zhang, Y. Wang, and D. Yang, "Ccspan: Mining closed contiguous sequential patterns," *Knowl.-Based Syst.*, vol. 89, pp. 1–13, 2015.

[11] M. J. Zaki, "Sequence mining in categorical domains: Incorporating constraints," in *CIKM*. ACM, 2000, pp. 422–429.

[12] E. Coquery, S. Jabbour, L. Saïs, and Y. Salhi, "A sat-based approach for discovering frequent, closed and maximal patterns in a sequence," in *ECAI*, ser. Frontiers in Artificial Intelligence and Applications, vol. 242. IOS Press, 2012, pp. 258–263.

[13] B. Négrevergne and T. Guns, "Constraint-based sequence mining using constraint programming," in *CPAIOR*, ser. Lecture Notes in Computer Science, vol. 9075. Springer, 2015, pp. 288–305.

[14] A. Kemmar, S. Loudni, Y. Lebbah, P. Boizumault, and T. Charnois, "PREFIX-PROJECTION global constraint for sequential pattern mining," in *CP*, ser. Lecture Notes in Computer Science, vol. 9255. Springer, 2015, pp. 226–243.

[15] ——, "A global constraint for mining sequential patterns with GAP constraint," in *CPAIOR*, ser. Lecture Notes in Computer Science, vol. 9676. Springer, 2016, pp. 198–215.

[16] J. O. R. Aoga, T. Guns, and P. Schaus, "Mining time-constrained sequential patterns with constraint programming," *Constraints*, vol. 22, no. 4, pp. 548–570, 2017.

[17] L. Dehaspe and H. Toivonen, "Discovery of frequent DATALOG patterns," *Data Min. Knowl. Discov.*, vol. 3, no. 1, pp. 7–36, 1999.

[18] F. Esposito, N. D. Mauro, T. M. A. Basile, and S. Ferilli, "Multi-dimensional relational sequence mining," *Fundam. Inform.*, vol. 89, no. 1, pp. 23–43, 2008.

[19] B. Cule and B. Goethals, "Mining association rules in long sequences," in *PAKDD (1)*, ser. Lecture Notes in Computer Science, vol. 6118. Springer, 2010, pp. 300–309.

[20] H. Cheng, X. Yan, J. Han, and C. Hsu, "Discriminative frequent pattern analysis for effective classification," in *ICDE*. IEEE Computer Society, 2007, pp. 716–725.

[21] C. Zhou, B. Cule, and B. Goethals, "Pattern based sequence classification," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 5, pp. 1285–1298, 2016.

[22] D. Fradkin and F. Mörchen, "Mining sequential patterns for classification," *Knowl. Inf. Syst.*, vol. 45, no. 3, pp. 731–749, 2015.

[23] J. M. Fowkes and C. A. Sutton, "A subsequence interleaving model for sequential pattern mining," in *KDD*. ACM, 2016, pp. 835–844.

[24] E. Egho, D. Gay, M. Boullé, N. Voisine, and F. Clérot, "A parameter-free approach for mining robust sequential classification rules," in *ICDM*. IEEE Computer Society, 2015, pp. 745–750.

[25] S. Lo, "Binary prediction based on weighted sequential mining method," in *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE Computer Society, 2005, pp. 755–761.

[26] J. H. Chang, "Mining weighted sequential patterns in a sequence database with a time-interval weight," *Knowl.-Based Syst.*, vol. 24, no. 1, pp. 1–9, 2011.

[27] F. Xie, X. Wu, and X. Zhu, "Efficient sequential pattern mining with wildcards for keyphrase extraction," *Knowl.-Based Syst.*, vol. 115, pp. 27–39, 2017.

[28] C. Di Ciccio, F. M. Maggi, and J. Mendling, "Efficient discovery of target-branched declare constraints," *Inf. Syst.*, vol. 56, pp. 258–283, 2016.

[29] F. Mourão, L. C. da Rocha, R. B. Araújo, T. Couto, M. A. Gonçalves, and W. M. Jr., "Understanding temporal aspects in document classification," in *WSDM*. ACM, 2008, pp. 159–170.

[30] G. Lebanon and Y. Zhao, "Local likelihood modeling of temporal text streams," in *ICML*, ser. ACM International Conference Proceeding Series, vol. 307. ACM, 2008, pp. 552–559.

[31] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst, "DECLARE: full support for loosely-structured processes," in *EDOC*. IEEE Computer Society, 2007, pp. 287–300.

[32] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *ICSE*. ACM, 1999, pp. 411–420.

[33] M. Pesić, "Constraint-based work on management systems: Shifting control to users," Ph.D. dissertation, PhD thesis, Eindhoven University of Technology, 2008. 26.

[34] M. Westergaard, C. Stahl, and H. A. Reijers, "Unconstrainedminer: efficient discovery of generalized declarative process models," *BPM Center Report BPM-13-28, BPMcenter. org*, p. 28, 2013.

[35] C. Di Ciccio and M. Mecella, "A two-step fast algorithm for the automated discovery of declarative workflows," in *CIDM*. IEEE, 2013, pp. 135–142.

[36] F. M. Maggi, R. P. J. C. Bose, and W. M. P. van der Aalst, "Efficient discovery of understandable declarative process models from event logs," in *CAiSE*, ser. Lecture Notes in Computer Science, vol. 7328. Springer, 2012, pp. 270–285.

[37] F. M. Maggi, M. Montali, C. Di Ciccio, and J. Mendling, "Semantical vacuity detection in declarative process mining," in *BPM*, ser. Lecture Notes in Computer Science, vol. 9850. Springer, 2016, pp. 158–175.

[38] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, 2014.

[39] H. T. Lam, F. Mörchen, D. Fradkin, and T. Calders, "Mining compressing sequential patterns," *Statistical Analysis and Data Mining*, vol. 7, no. 1, pp. 34–52, 2014.

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[41] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu, and V. S. Tseng, "SPMF: a java open-source pattern mining library," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3389–3393, 2014.

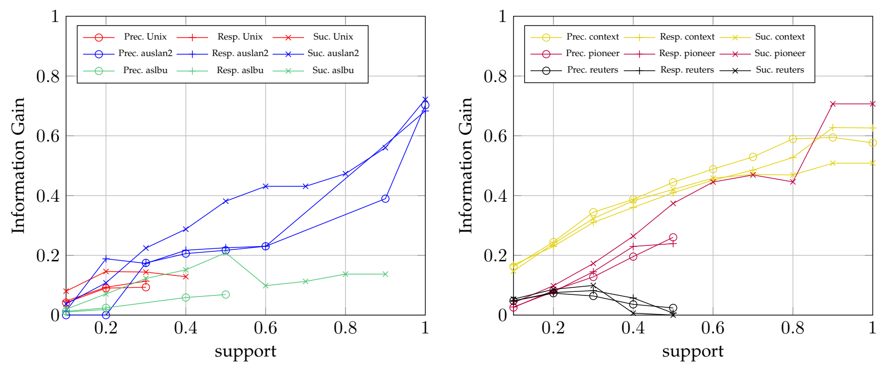
[42] J. T. Kent, "Information gain and a general measure of correlation," *Biometrika*, vol. 70, no. 1, pp. 163–173, 1983.

[43] F. M. Maggi, M. Dumas, L. García-Bañuelos, and M. Montali, "Discovering data-aware declarative process models from event logs," in *BPM*, ser. Lecture Notes in Computer Science, vol. 8094. Springer, 2013, pp. 81–96.

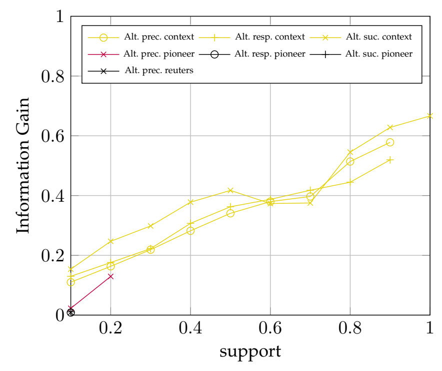
Johannes De Smedt is currently a lecturer in business analytics at the University of Edinburgh Business School. His research interests include constraint-based process modeling and mining, ontologies in processes, and sequence mining. He received his PhD in Information Systems Engineering at KU Leuven and his work has been published in well-known information systems engineering and machine learning conferences and journals.

Galina Deeva is a PhD researcher at the Department of Decision Sciences and Information Management of KU Leuven. She obtained her MSc in Management Science at Harbin Institute of Technology, China. Her research interests include sequence and process mining, machine learning and learning analytics.

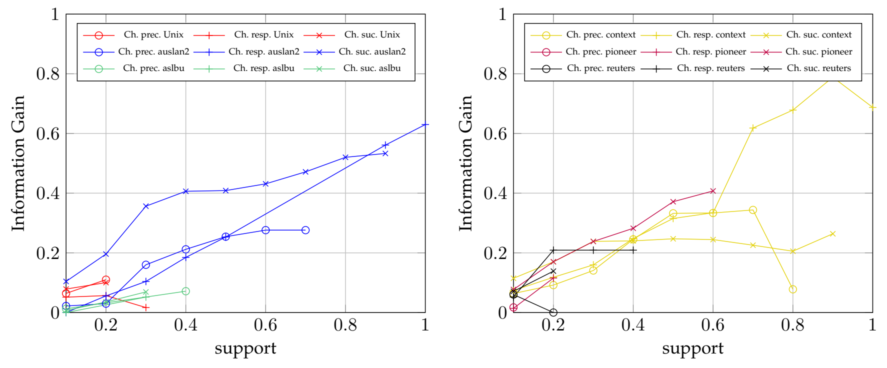
Jochen De Weerdt is an assistant professor at the Department of Decision Sciences and Information Management of KU Leuven. He received his PhD in Business Economics at KU Leuven and worked as a postdoctoral research fellow at the Information Systems School of Queensland University of Technology. His research expertise is mainly in process mining, data analytics, and business process management. His findings have been published in well-known international journals and conferences.



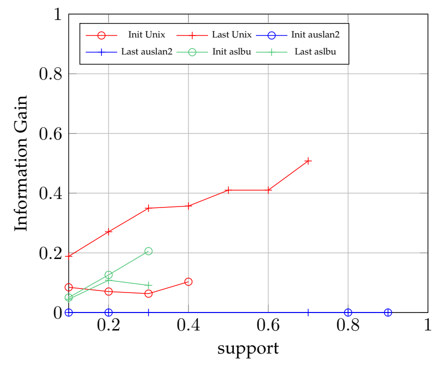
(a) Simple ordered



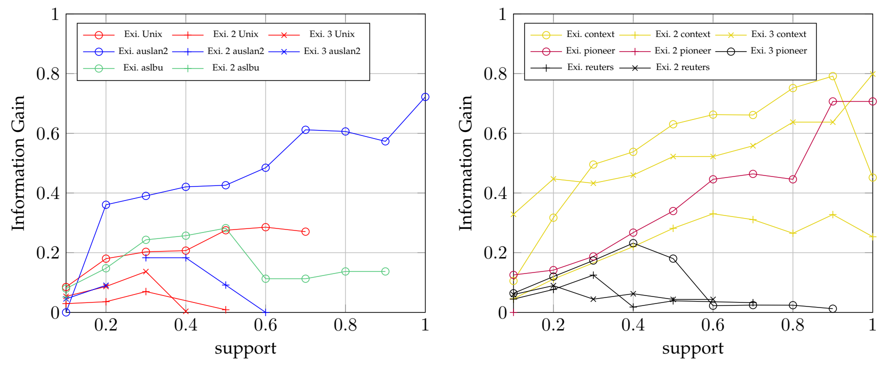
(b) Alternating ordered



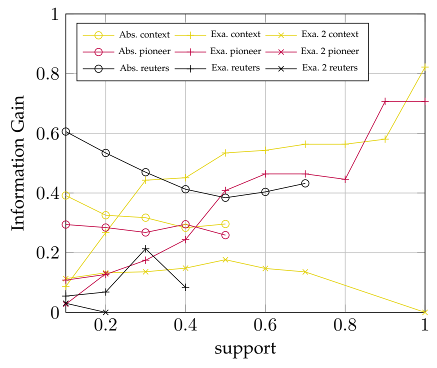
(c) Chain ordered



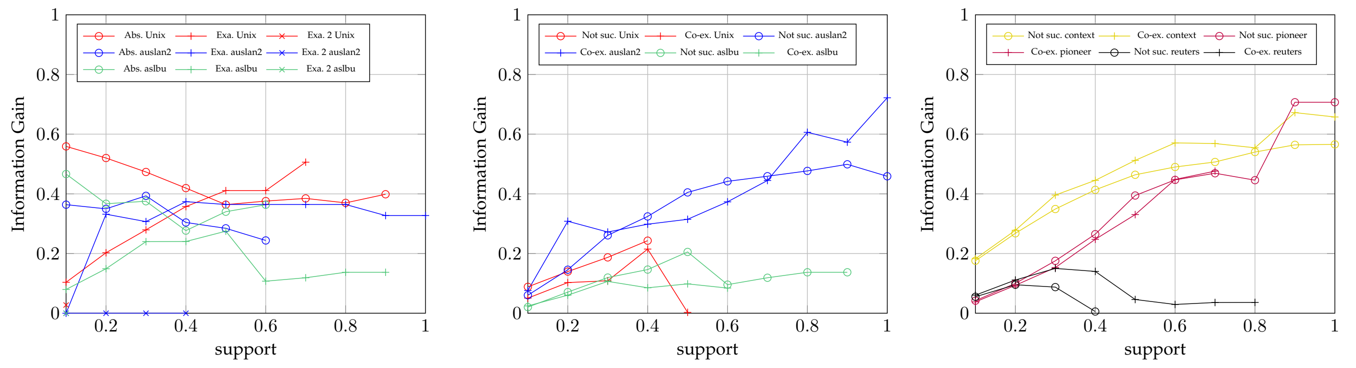
(d) Position



(e) Existence



(f) Absence and exactly (1)



(g) Absence and exactly (2) and not succession and co-existence

Figure 2: Overview of the information gain of the constraints per dataset.