Behavioral constraint template-based sequence classification

Behavioral Constraint Template-Based Sequence Classification

Johannes De Smedt*[1], Galina Deeva[2], Jochen De Weerdt[2]

[1] University of Edinburgh, Business School, Management Science and Business Economics Group
[2] KU Leuven, Faculty of Economics and Business, Department of Decision Sciences and Information Management

Abstract. In this paper we present the interesting Behavioral Constraint Miner (iBCM), a new approach towards classifying sequences. The prevalence of sequential data, i.e., a collection of ordered items such as text, website navigation patterns, traffic management, and so on, has incited a surge in research interest towards sequence classification. Existing approaches mainly focus on retrieving sequences of itemsets and checking their presence in labeled data streams to obtain a classifier. The proposed iBCM approach, rather than focusing on plain sequences, is template-based and draws its inspiration from behavioral patterns used for software verification. These patterns have a broad range of characteristics and go beyond the typical sequence mining representation, allowing for a more precise and concise way of capturing sequential information in a database. Furthermore, it is possible to also mine for negative information, i.e., sequences that do not occur. The technique is benchmarked against other state-of-the-art approaches and exhibits a strong potential towards sequence classification.

Keywords. Sequence Mining, Sequence Classification, Constraint-Based Mining

1 Introduction

Analyzing sequential data [1] has seen a vast surge in interest during recent years, driven by the growth of typical sources such as DNA databases, text repositories, road analysis [2] and user behavior analysis [3]. Many techniques exist to derive ordered items from temporal databases, focusing on either different techniques for discovery, e.g., using prefix-oriented and constraint-based approaches, or towards different outcomes, e.g., regular expressions or closed sequences. These sequential features can be used for classifying new database entries, a discipline that does not only focus on constructing the most complete set of features, but rather the most discriminating.

In this paper, we propose a new sequence classification technique, called iBCM (interesting Behavioural Constraint Miner), which featurizes sequences

* Corresponding author.

according to a predefined set of behavioral constraint templates. As such, a fine-granular view of the temporal relations between items can be achieved and applied towards classification. Furthermore, iBCM allows for easy identification of the differences between classes, and gives insight into what types of relations are typically relevant for classification. In the experimental evaluation, it is shown that iBCM is capable of obtaining high discriminative power while minimizing the number of features needed. In addition, only deriving a certain type of constraint templates can already capture the most discriminating features.

This paper is structured as follows. In Section 2, an overview of the state-of-the-art of both sequence mining and classification is discussed. In Section 3, the backdrop for mining behavioral sequence patterns is introduced, which leads into the discussion of the inference part of iBCM in Section 4. Next, Section 5 reports on a benchmark with other state-of-the-art techniques. Finally, Section 6 summarizes the contributions and provides suggestions for future work.

2 State-of-the-art

In this section, an overview of existing sequence mining and classification techniques is discussed.

2.1 Sequence Mining

Sequence mining, also referred to as frequent itemset mining or temporal data mining, has been tackled in numerous ways. The original approach was rooted in frequent itemset discovery [4] and based on apriori-concepts. Extensions to this original approach have been proposed to obtain closed sequences [5] and to achieve performance benefits through prefix representation of the dataset [6]. A constraint-based approach was proposed in [7] in the form of cSPADE, and has recently seen a strong interest towards extending it along the declarative constraint programming paradigm. More specifically, several studies investigate how to generically build a knowledge base of constraints covering the sequences in a temporal dataset. For example, in [8], a satisfiability-based technique is devised for enumerating all frequent sequences using cardinalities for the constraints retrieved. In [9] a better prefix representation for sequences mining constraints was introduced, which was later extended for GAP constraints [10]. A similar approach was devised in [11], in which the authors propose 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 can also be used to quickly retrieve regular expressions. Finally, in [12], a general constraint programming approach that steers away from explicit wildcards is introduced.

2.2 Sequence Classification

While many insights from sequence mining carry over into sequence classification, the nature of the objective is different. Rather than eliciting the full set of

sequences or constraints supported, it is paramount that the feature set exhibits the following characteristics.

- **Compact:** in order to build classifiers in reasonable time, the set of features should be reduced to a minimum,
- **Interesting:** 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 [13]. In general, there is a need for a balance in the feature set that strikes support values in between extremely high and low values [14],
- **Concise:** the feature set is small though comprehensive, and explains the sequential behavior in an understandable way.

Many sequence classification techniques have been proposed [15,16,17], each focusing on a different approach ranging from extensions to sequence pattern mining algorithms, to statistical approaches that infer the explanatory power of subsequences. They can be classified as either being direct, i.e., the features are extracted according to their strength towards the classifier, or indirect, i.e., all features are generated and later selected by a classifier. [15] extends the SPADE algorithm with an interestingness measure that is based on both the support and the window (cohesion) in which the items of the constraint occur. In [16] BIDE-D(C) is introduced which rather incorporates information gain into BIDE to provide a direct sequence classification approach. In [17], 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 are able to reconstruct the sequence database. A similar approach is used in [18], where a strong emphasis is used towards finding interesting sequences.

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 elicit the sequence database as a whole, highly diverse and informative behavioral patterns are used that incorporate cardinality, alteration, gaps, as well as negative information. By fixing the pattern base, it becomes easy to write a specific and fast algorithm for retrieving them from large databases. The technique employs only binary constraints, however, other studies such as [15] have already revealed that for sequence classification, the length of the patterns does not have to exceed 3, or even 2.

3 The Framework of Behavioral Templates

In this section, the preliminaries are established and an overview of the behavioral constraint templates/patterns and their characteristics is given.

3.1 Sequences and Sequence Databases

The task of sequence classification relies on the principles of both a sequence and a sequence database, as well as the classes or labels needed to discern their behavior.

Definition 1. *A sequence $\sigma = \langle \sigma_1, \sigma_2, ..., \sigma_n \rangle$ is a list of items with length $|\sigma| = n$ out of the alphabet Σ_σ. We denote:*

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

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

Definition 2. *A sequence database $\mathcal{SB}$ is a set of sequences with $L_{\mathcal{SB}}(\sigma) \to \mathbb{N}$ a labeling function assigning a class label to a sequence consisting of the items in $\Sigma_{\mathcal{SB}}$. The number of sequences in the database is $|\mathcal{SB}|$.*

Consider the example sequence database in Table 1, with $\Sigma_{\mathcal{SB}} = \{a, b, c\}$, $|\mathcal{SB}| = 6$, and $|L_{\mathcal{SB}}| = 2$.

Table 1: Example 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

3.2 Declare Pattern Base

The iBCM approach relies on a set of behavioral constraint templates based on the Declare language [19], which itself is inspired by the formal verification patterns of Dwyer [20]. These 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 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. The choice constraint can be considered an *existence* constraint over multiple items. The binary constraints exhibit a hierarchy [23]. There are unordered constraints (*responded/co-existence*), simple ordered (*precedence, response, succession*), alternating ordered, and chain ordered constraints. Hence, the opportunity exists to express not only the ordering, but also the repeating (alternation) and local (chain) behavior of two items. Furthermore, there are negative constraints, expressing behavior that does not occur. These can prove especially useful in the context of classification, and are typically not generated by sequence classification techniques that only mine for positive patterns.

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

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

Template	**LTL Formula [21]**	**Regular Expression [22]**
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
Responded existence(A,B)	$\Diamond A \implies \Diamond B$	[^A]*((A.*B.*) |(B.*A.*))?
Co-existence(A,B)	$\Diamond A \Longleftrightarrow \Diamond B$	[^AB]*((A.*B.*) |(B.*A.*))?
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]*
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 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)	$\Box(A \Longleftrightarrow \bigcirc B)$	[^AB]*(AB[^AB]*)*
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(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]*)

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 type 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 with a certain regular expression which can be converted into an FSM. We denote the corresponding regular expression as $\S(t)$. We write the FSM $\mathcal{A}$ corresponding with the regular expression as $\mathcal{A} = \S(A, t)$ or $\mathcal{A} = \S(altPrec(a, b))$. An example of $altPrec(a, b)$ is depicted in Figure 1.

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 Comparison with Other Sequence Constraint Representation

The iBCM approach does not intend to be able to reproduce the database. Rather it is able to capture the most discerning sequence-based features. Consider for example the database in Table 1. Table 3 lists the constraints that are

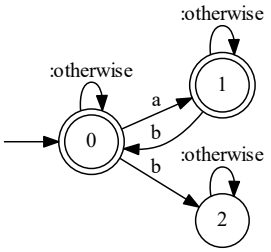


Fig. 1: Automaton of *alternate precedence(a,b)*.

present for both labels. Notice that for label 1, a does not always precede b. Also, for label 2 c occurs before b. This can be discerned by only 3 constraints which are marked in bold. Hence, with only 3 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 hierarchy between the constraints. To achieve the

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)

same results with typical sequence-based constraints as used in, e.g., SPADE, it is harder to make such concise distinctions, as non-local information present in, e.g., *succession* requires either longer or more sequences to approach the behavior that will converge towards the language of the regular expression.

To summarize, iBCM exhibits the following advantages:

- It employs a rich, varied set of constraints 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 can be converted into a global automaton for representing behavior graphically.

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 part are separate. Section 5 outlines the performance of the constraints generated by the approach as binary features (present/not present).

4.1 Inferring Constraints

The featurization approach is employed as a 3-step approach.

Step 1: Retain frequent items First, only items that exceed the support threshold are withheld in set A. Only these items will be used for checking unary constraints, and will be used in pair for checking binary constraints.

Step 2: Generate constraints Next, every sequence in the database is checked in the following manner. The sequence is traversed completely, and for every item in the frequent itemset the positions are stored. This allows for easy verification of the binary constraints. For every item $a \in A$, $|occ(a, \sigma|$ is used for determining the cardinality constraints, i.e., absence/exactly/existence. It is also checked whether it occurred as the first or last item in the sequence. Next, a is paired with every other $b \in A \setminus a$ to determine the type of behavioral constraint pattern. If a happens before b, the precedence lineage is reviewed. 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*. Both checks stop when there is no further evidence. If all occurrences of b fit, the constraints are added to the constraint set. If b happens after a, the response hierarchy is scrutinized. 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*.

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. This allows for precise measuring of sequential behavior, as some sequences might support both *response* and *precedence*, and others do not. Still, they can be merged (i.e. the simultaneous presence of *response* and *precedence* forms *succession*) to reduce the size of the number of features.

As can be seen from the algorithm, the binary constraints can be derived very efficiently by boolean and string operations. The approach is inspired by both [24] and [23]. However, for classification purposes the sequences need to be labeled right away. The former uses DFAs 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. The latter builds a knowledge base of occurrence and precedence relations and calculates the support for constraints. This, however, is done on a log level, rather than at entry/sequence level, which requires extra featurization steps afterwards.

Algorithm 1 Mining constraint features per class i

procedure RETRIEVE_CONSTRAINTS($\mathcal{SB}, minsup$) ▷ Input: Data and parameters
  $A \leftarrow frequentItems(\Sigma_{\mathcal{SB}}, minsup)$ ▷ Retain only frequent items
  for $\sigma \in \mathcal{SB}$ **do**
    $C_{L(\sigma)} \leftarrow mineConstraints(\sigma, A)$
  for $c \in C_{L(\sigma)}$ **do**
    if $sup(c) \geq |C_{L(\sigma)}| \times minsup$ **then**
      $C_{\mathcal{SB}} \leftarrow c$
  applyHierarchyReduction
  return $C_{\mathcal{SB}}$

4.2 Considerations on Constraint Template Base

Not all Declare constraint templates are fit 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* 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 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 of not 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 they are not included in the final constraint set. *Not succession* is the only negative constraint used. Note that all constraints are mined with a confidence of 100%.

Algorithm 2 Mining behavioral constraint templates

```
1: procedure MINECONSTRAINTS(σ, A)
2:     C ← ∅
3:     for σ_i ∈ σ do occ(σ_i, σ) ← i
4:     for a ∈ A ∩ Σ_σ do
5:         if |occ(a, σ)| = 0 then C ← absence(a, 1)                    ▷ Unary constraints
6:         else if |occ(a, σ)| > 2 then C ← existence(a, 3)
7:         else C ← exactly(a, |occ(a, σ)|)
8:         if 1 ∈ occ(a, σ) then C ← init(a)
9:         if |σ| ∈ occ(a, σ) then C ← last(a)
10:        for b ∈ A ∩ Σ_σ do                                           ▷ Binary constraints
11:            if min(occ(a, σ)) < min(occ(b, σ)) then
12:                C ← prec(a, b)
13:                i ← min(occ(b, σ)), chain ← (i − 1) ∈ occ(a, σ), continue ← ⊤
14:                while ∃n ∈ occ(b, σ), n > i ∧ continue do
15:                    if ∃p ∈ occ(a, σ), i < p < n then i ← n
16:                        if ¬chain ∨ (n − 1) ∉ occ(a, σ) then chain ← ¬
17:                    else continue ← ¬
18:                if continue then C ← altPrec(a, b)
19:                if chain then C ← chainPrec(a, b)
20:            if max(occ(a, σ)) < max(occ(b, σ)) then
21:                C ← resp(a, b)
22:                if max(occ(a, σ)) < min(occ(b, σ)) then C ← notSuc(a, b)
23:                i ← min(occ(a, σ)), chain ← (i + 1) ∈ occ(b, σ), continue ← ⊤
24:                while ∃n ∈ occ(a, σ), n > i ∧ continue do
25:                    if ∃p ∈ occ(b, σ), i < p < n then C ← altResp(a, b), i ← n
26:                        if ¬chain ∨ (n + 1) ∉ occ(b, σ) then chain ← ¬
27:                    else continue ← ¬
28:                if continue then C ← altResp(a, b)
29:                if chain then C ← chainResp(a, b)
30:            add succession if (alternate/chain) response and precedence
31:            if b ∉ Σ_σ ∧ b ∈ A then
32:                C ← exclChoi(a, b)
33:    return C
```

4.3 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 they are 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 $O(|A|^2 \times |\sigma|)$.

As will become clear from experimental evaluation, however, iBCM can achieve good results at high minimum support levels, reducing $|A|$ drastically.

5 Experimental Evaluation

In this section, the technique will be evaluated on widely-used, realistic datasets and compared with 4 other approaches.

5.1 Setup

Below, an overview of the data, implementation, and other approaches is given.

Data and Classification The datasets that were used are summarized in Table 4 and are a mix with both a large set of distinct items, as well as a large number of data entries. They are discussed in more detail in [17] and [15]. All

Table 4: Characteristics of the datalogs used for evaluation.

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

techniques were first employed to generate interesting sequences, and next to build a predictive model by using the presence of the sequences as a binary feature. Three classifiers were considered, i.e., naive Bayes (NB), decision trees (DT), and random forests (RF), for which the Weka[3] Java implementation was used. All runs were executed using a Java 8 Virtual Machine on an Intel i7-6700HQ CPU with 16GB DDR4 memory. A 10-fold cross-validation was applied for all the experiments.

Approaches iBCM is benchmarked against 4 other state-of-the-art techniques, being cSPADE [25], Interesting Sequence Miner (ISM) [17], Sequence Classification based on Interesting Sequences (SCIP) [15], and Mining Sequential Classification Rules (MiSeRe) [18], which all have the clear goal of obtaining discriminative, informative sequences for classification and are compared in Table 5. A comparison with other techniques can be found in the respective works as well. For cSPADE, 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. For MiSeRe, 1, 2, 5, and 10 second run times were considered. Finally ISM was used 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. The implementation of the benchmark can be found online at `http://feb.kuleuven.be/public/n13076/`.

5.2 Results

The results in terms of accuracy and the number of generated constraints along the support spectrum are displayed in Figures 2 and 3. The results for ISM and

[3] `http://www.cs.waikato.ac.nz/ml/weka/`

Table 5: An overview of the techniques used for benchmarking.

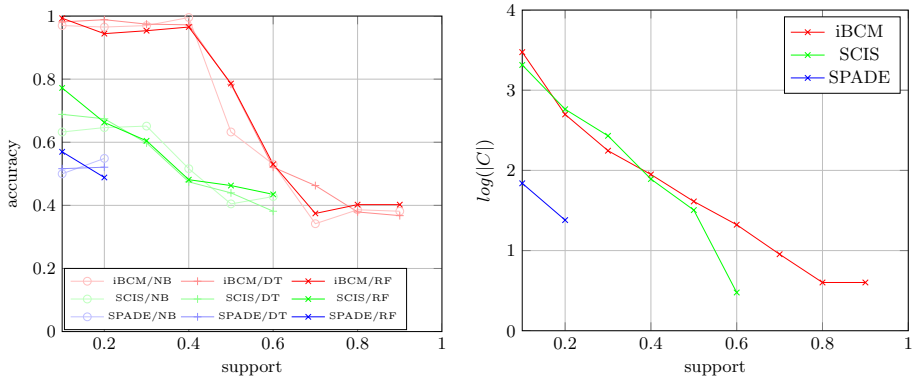
Technique	Description	Parameters
cSPADE [7]	Sequence mining approach based on window, gap, length, width, and other constraints.	Support
SCIP [15]	Extension to SPADE based on an interesting measure that next to the support of a sequence also consists of the proximity of its items.	Support, interestingness, maximum length of sequences
ISM [17]	Technique that interleaves subsequences and infers the most compact set of sequences that can regenerate the database.	max # optimization steps, max # iterations
MiSeRe [18]	Randomly generates diverse sequences and applies a Bayesian approach to retain interesting sequences.	Maximum runtime (in seconds)
iBCM	The devised approach, based on mining a set of behavioral constraints.	Support

MiSeRe are reported separately in Table 6. An overview of the share of each constraint template family in the results of iBCM is given in Table 4.

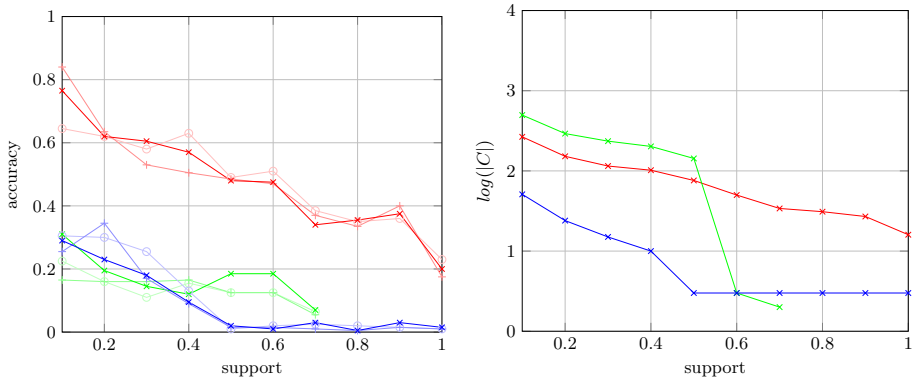
Overall, iBCM is capable of achieving a high accuracy, without inducing a big amount of constraints ($|C|$). Especially for the *aslbu* and *auslan* datasets, a higher accuracy is obtained than using the state-of-the-art techniques. Also, iBCM achieves a higher accuracy more rapidly when going down the support spectrum, achieving high accuracy already for 50% to 60% with a small amount of constraints (<100). The differences in terms of the type of sequential behavior present becomes apparent. In the text-based datasets, such as *news*, the *absence* constraint clearly provides a prominent source of information, since rather the presence of items, not relations, are needed for classification. This lies in line with the findings in [17]. In the other datasets, the whole set of constraint patterns is used, except for the very specific *chain* constraints. The inclusion of negative constraints might explain the higher accuracy for *aslbu* and *auslan2*. The more comprehensive *alternating* constraints are indeed often present (note that the hierarchy reduction cuts away all simple/alternating ordered constraints when alternating/chain constraints are found).

cSPADE was not able to finish executing the *context* dataset within 60 minutes for support values lower than 50%. Similarly, ISM was not able to do generate interesting sequences for the *News* dataset. Besides, the algorithms did not always generate constraints for certain higher support values. In terms of performance, in Figure 5 the time needed to generate the constraints and label the sequences is plotted. All constraints could be derived in less than 1 second, except for the News dataset due to the bigger size of $|A|$. In this case, the technique clearly scales exponentially with the size of $|A|$. This is probably due to the nature of the data, being plain text. The higher number of items, of which there are no particularly frequent after a certain threshold, increases the runtime. In the other datasets, infrequent items are truly infrequent and $|A|$ does not necessarily grow. Nevertheless, support settings as of 0.6 already guarantee a decent level of accuracy.

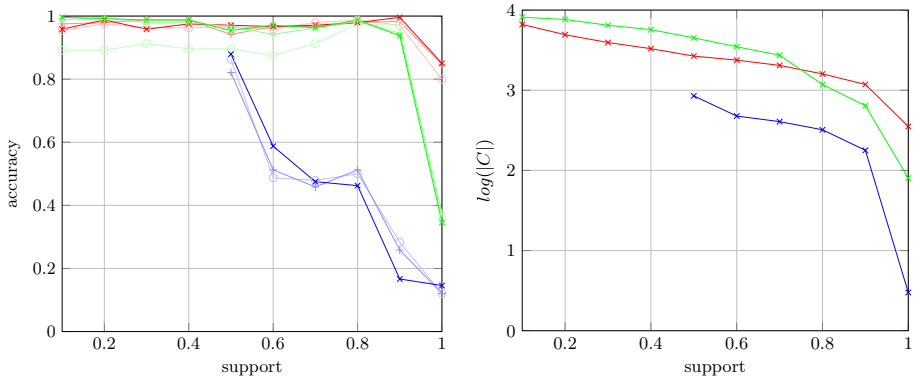
There is no notable difference in accuracy when using different classifiers, except for the *aslbu* and *auslan2* datasets. Especially the constraints generated



(a) aslbu

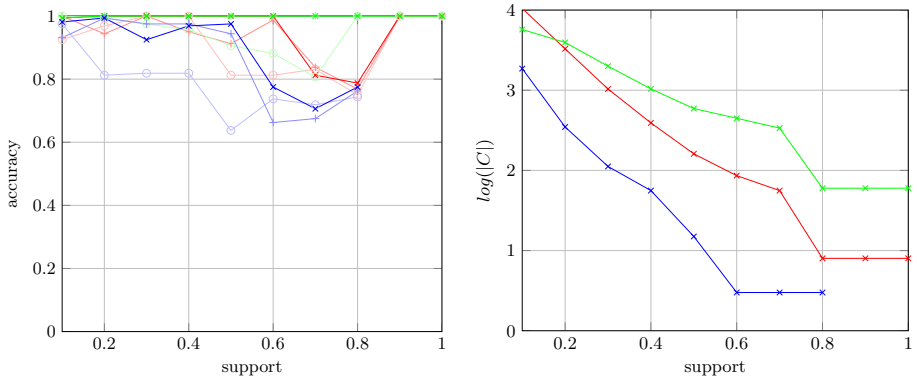


(b) auslan2

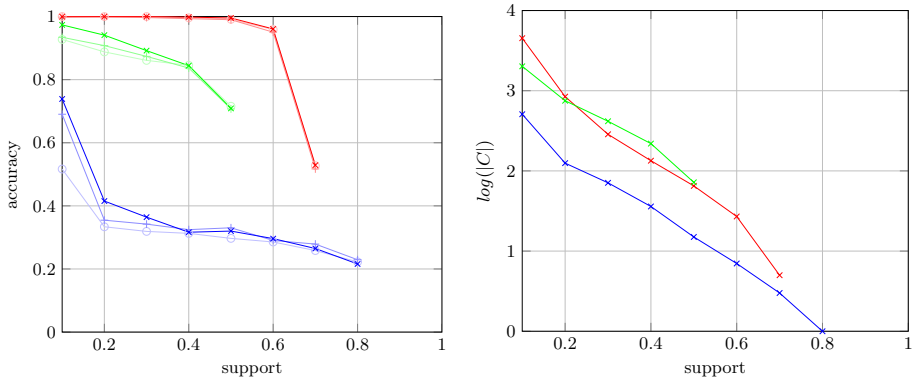


(c) context

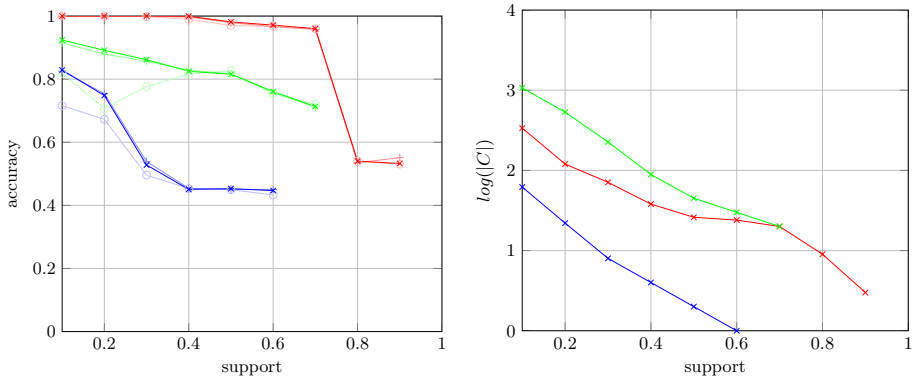
Fig. 2: Overview of the performance of the different algorithms.



(a) pioneer



(b) News



(c) Unix

Fig. 3: Overview of the performance of the different algorithms.

Table 6: Accuracny and log($|C|$) (between brackets) results for MiSeRe and ISM.

Dataset	Classifier	misere (1s)	misere (2s)	misere (5s)	misere (10s)	ISM
aslbu	**NB**	0.56 (2.086)	0.556 (2.111)	0.542 (2.111)	**0.595 (2.107)**	0.602 (2.274)
	DT	0.542 (2.083)	0.547 (2.1)	0.556 (2.111)	0.544 (2.111)	0.626 (2.274)
	RF	0.53 (2.097)	0.593 (2.111)	0.586 (2.111)	0.595 (2.111)	**0.623 (2.274)**
auslan2	**NB**	**0.37 (2.4)**	0.31 (2.401)	0.37 (2.401)	0.26 (2.401)	0.225 (1.279)
	DT	0.27 (2.401)	0.305 (2.401)	0.27 (2.401)	0.25 (2.401)	**0.24 (1.279)**
	RF	0.25 (2.401)	0.325 (2.401)	0.305 (2.401)	0.295 (2.401)	**0.24 (1.279)**
context	**NB**	**0.938 (2.587)**	0.938 (2.892)	0.929 (3.259)	0.888 (3.568)	**0.821 (2.025)**
	DT	0.888 (2.592)	0.871 (2.854)	0.908 (3.282)	0.867 (3.547)	**0.821 (2.025)**
	RF	0.908 (2.645)	0.888 (2.814)	0.933 (3.235)	0.913 (3.554)	0.725 (2.025)
pioneer	**NB**	0.963 (1.903)	0.969 (1.982)	0.813 (2.201)	0.863 (2.369)	0.981 (2.093)
	DT	0.994 (1.845)	0.988 (1.959)	0.988 (2.152)	0.988 (2.336)	0.963 (2.093)
	RF	0.994 (1.863)	0.994 (2.021)	**1 (2.188)**	1 (2.342)	**1 (2.093)**
Unix	**NB**	0.863 (2.423)	0.86 (2.447)	0.815 (2.687)	0.744 (2.707)	0.897 (3.232)
	DT	0.901 (2.42)	0.902 (2.616)	0.898 (2.549)	0.903 (2.702)	**0.927 (3.232)**
	RF	**0.923 (2.511)**	0.91 (2.555)	0.912 (2.716)	0.91 (2.842)	0.908 (3.232)
News	**NB**	0.929 (3.445)	0.929 (3.445)	0.928 (3.446)	0.919 (3.448)	NA
	DT	0.899 (3.446)	0.901 (3.446)	0.902 (3.446)	0.905 (3.447)	NA
	RF	**0.973 (3.445)**	0.969 (3.445)	0.971 (3.446)	0.971 (3.445)	NA

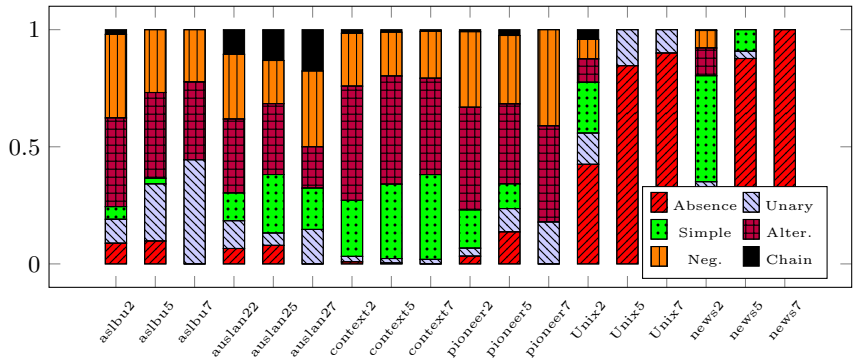


Fig. 4: An overview of the share (in %) of the type of constraint templates mined from the databases. The numbers stand for the minimum support, e.g., 2 stands for 20%.

by cSPADE seem to have a different impact on the classifiers. In general, the classifiers perform more stably on the datasets with either less labels or with more sequences to learn from. Random forests seem to perform the best overall.

6 Conclusion and Future Work

This work proposed iBCM, a new technique with the ability to discover features for sequence classification. Based on behavioral constraint templates, iBCM is able to concisely distinguish different sequential behaviors in databases. It is

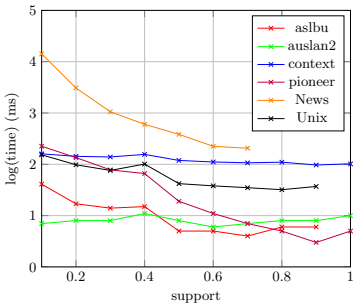


Fig. 5: Time needed to mine constraints.

capable of achieving results with high accuracy, while minimizing the number of features needed compared with other approaches. Furthermore, the inference technique devised can also be applied towards descriptively interpreting the nature of the patterns present in a sequence database, offering insights into what types of interplay are present between the items in the data.

In future work, a more in-depth comparison of which types of constraints contribute the most to the classifiers will be made. This establishes the base for building direct sequence classification techniques as well. Finally, the data-aware versions of the patterns can be introduced as well. Most patterns are also described in first-order LTL and can be extended to include non-sequential information [26]. Also, the target-branched version of Declare [27], i.e., constraints with a consequent being a set rather than a singleton will be investigated.

References

1. Agrawal, R., Srikant, R.: Mining sequential patterns. In: ICDE, IEEE Computer Society (1995) 3–14
2. Lee, J., Han, J., Li, X., Cheng, H.: Mining discriminative patterns for classifying trajectories on road networks. IEEE Trans. Knowl. Data Eng. **23**(5) (2011) 713–726
3. Eichinger, F., Nauck, D.D., Klawonn, F.: Sequence mining for customer behaviour predictions in telecommunications. In: Proceedings of the Workshop on Practical Data Mining at ECML/PKDD. (2006) 3–10
4. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: SIGMOD Conference, ACM Press (1993) 207–216
5. Wang, J., Han, J.: BIDE: efficient mining of frequent closed sequences. In: ICDE, IEEE Computer Society (2004) 79–90
6. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Prefixspan: Mining sequential patterns by prefix-projected growth. In: ICDE, IEEE Computer Society (2001) 215–224
7. Zaki, M.J.: SPADE: an efficient algorithm for mining frequent sequences. Machine Learning **42**(1/2) (2001) 31–60

8. Coquery, E., Jabbour, S., Saïs, L., Salhi, Y.: A sat-based approach for discovering frequent, closed and maximal patterns in a sequence. In: ECAI. Volume 242 of Frontiers in Artificial Intelligence and Applications., IOS Press (2012) 258–263
9. Kemmar, A., Loudni, S., Lebbah, Y., Boizumault, P., Charnois, T.: PREFIX-PROJECTION global constraint for sequential pattern mining. In: CP. Volume 9255 of Lecture Notes in Computer Science., Springer (2015) 226–243
10. Kemmar, A., Loudni, S., Lebbah, Y., Boizumault, P., Charnois, T.: A global constraint for mining sequential patterns with GAP constraint. In: CPAIOR. Volume 9676 of Lecture Notes in Computer Science., Springer (2016) 198–215
11. Aoga, J.O.R., Guns, T., Schaus, P.: An efficient algorithm for mining frequent sequence with constraint programming. In: ECML/PKDD (2). Volume 9852 of Lecture Notes in Computer Science., Springer (2016) 315–330
12. Négrevergne, B., Guns, T.: Constraint-based sequence mining using constraint programming. In: CPAIOR. Volume 9075 of Lecture Notes in Computer Science., Springer (2015) 288–305
13. Cule, B., Goethals, B.: Mining association rules in long sequences. In: PAKDD (1). Volume 6118 of Lecture Notes in Computer Science., Springer (2010) 300–309
14. Cheng, H., Yan, X., Han, J., Hsu, C.: Discriminative frequent pattern analysis for effective classification. In: ICDE, IEEE Computer Society (2007) 716–725
15. Zhou, C., Cule, B., Goethals, B.: Pattern based sequence classification. IEEE Trans. Knowl. Data Eng. **28**(5) (2016) 1285–1298
16. Fradkin, D., Mörchen, F.: Mining sequential patterns for classification. Knowl. Inf. Syst. **45**(3) (2015) 731–749
17. Fowkes, J.M., Sutton, C.A.: A subsequence interleaving model for sequential pattern mining. In: KDD, ACM (2016) 835–844
18. Egho, E., Gay, D., Boullé, M., Voisine, N., Clérot, F.: A parameter-free approach for mining robust sequential classification rules. In: ICDM, IEEE Computer Society (2015) 745–750
19. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: EDOC, IEEE Computer Society (2007) 287–300
20. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE, ACM (1999) 411–420
21. Pesić, M.: Constraint-Based Work on Management Systems: Shifting Control to Users. PhD thesis, PhD thesis, Eindhoven University of Technology, 2008. 26
22. Westergaard, M., Stahl, C., Reijers, H.A.: Unconstrainedminer: efficient discovery of generalized declarative process models. BPM Center Report BPM-13-28, BPMcenter. org (2013) 28
23. Di Ciccio, C., Mecella, M.: A two-step fast algorithm for the automated discovery of declarative workflows. In: CIDM, IEEE (2013) 135–142
24. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: CAiSE. Volume 7328 of Lecture Notes in Computer Science., Springer (2012) 270–285
25. Zaki, M.J.: Sequence mining in categorical domains: Incorporating constraints. In: CIKM, ACM (2000) 422–429
26. Maggi, F.M., Dumas, M., García-Bañuelos, L., Montali, M.: Discovering data-aware declarative process models from event logs. In: BPM. Volume 8094 of Lecture Notes in Computer Science., Springer (2013) 81–96
27. Di Ciccio, C., Maggi, F.M., Mendling, J.: Efficient discovery of target-branched declare constraints. Inf. Syst. **56** (2016) 258–283