SeqScout: Using a Bandit Model to Discover Interesting Subgroups in Labeled Sequences

Romain Mathonat, Diana Nurbakova, Jean-François Boulicaut, Mehdi Kaytoue

SeqScout: Using a Bandit Model to Discover Interesting Subgroups in Labeled Sequences

Romain Mathonat
1. Université de Lyon
CNRS, INSA-Lyon, LIRIS
UMR5205, F-69621
Villeurbanne, France
2. ATOS
F-69100 Villeurbanne, France
romain.mathonat@insa-lyon.fr

Diana Nurbakova
Université de Lyon
CNRS, INSA-Lyon, LIRIS
UMR5205, F-69621
Villeurbanne, France
diana.nurbakova@insa-lyon.fr

Jean-Francois Boulicaut
Université de Lyon
CNRS, INSA-Lyon, LIRIS
UMR5205, F-69621
Villeurbanne, France
jfboulicaut@gmail.com

Mehdi Kaytoue
1. Université de Lyon
CNRS, INSA-Lyon, LIRIS
UMR5205, F-69621
Villeurbanne, France
2. Infologic
F-69007 Lyon, France
mehdi.kaytoue@gmail.com

***Abstract*—It is extremely useful to exploit labeled datasets not only to learn models but also to improve our understanding of a domain and its available targeted classes. The so-called subgroup discovery task has been considered for a long time. It concerns the discovery of patterns or descriptions, the set of supporting objects of which have interesting properties, e.g., they characterize or discriminate a given target class. Though many subgroup discovery algorithms have been proposed for transactional data, discovering subgroups within labeled sequential data and thus searching for descriptions as sequential patterns has been much less studied. In that context, exhaustive exploration strategies can not be used for real-life applications and we have to look for heuristic approaches. We propose the algorithm `SeqScout` to discover interesting subgroups (w.r.t. a chosen quality measure) from labeled sequences of itemsets. This is a new sampling algorithm that mines discriminant sequential patterns using a multi-armed bandit model. It is an anytime algorithm that, for a given budget, finds a collection of local optima in the search space of descriptions and thus subgroups. It requires a light configuration and it is independent from the quality measure used for pattern scoring. Furthermore, it is fairly simple to implement. We provide qualitative and quantitative experiments on several datasets to illustrate its added-value.**

***Keywords*-Pattern Mining, Subgroup Discovery, Sequences, Upper Confidence Bound.**

I. Introduction

In many data science projects we have to process labeled data and it is often valuable to discover patterns that discriminate the class values. This can be used to support various machine learning techniques the goals of which are to predict class values for unseen objects. It is also interesting *per se* since the language for descriptions is, by design, readable and interpretable by analysts and data owners. Among others, it can support the construction of new relevant features. The search for such patterns has been given different names like subgroup discovery, emerging pattern or contrast set mining [1]. We use hereafter the terminology of the subgroup discovery framework [2].

Labeled sequential data are ubiquitous and subgroup discovery can be used throughout many application domains like, for instance, text or video data analysis, industrial process supervision, DNA sequence analysis, web usage mining, video game analytics, etc. Let us consider an example scenario about the maintenance of a cloud environment. Data generated from such a system are sequences of events. Applying classification techniques helps answering to "*will* an event occur" (See, e.g., [3]), while applying sequential event prediction helps determining "*what* event will occur" (See, e.g., [4]). Nevertheless, another need is to explain, or at least, to provide hypotheses on the *why*. Addressing such a *descriptive analytics* issue is the focus of our work. Given data sequences, labeled with classes, we aim at automatically finding discriminative patterns for these classes. Given our example, sequences of events could be labeled by breakdown presence or absence: the goal would be to compute patterns "obviously occurring with breakdowns". Such patterns provide valuable hypotheses for a better understanding of the targeted system and it can support its maintenance.

Let us now discuss research issues when considering subgroup discovery on labeled data. Given a dataset of object descriptions (e.g., objects described by discrete sequences), a sequential pattern is a generalization of a description that covers a set of objects. An unusual class distribution among the covered objects makes a pattern interesting. For example, when a dataset has a 99%-1% distribution of normal-abnormal objects, a pattern covering objects among which 50% are abnormal is highly relevant (w.r.t. a quality measure like the Weighted Relative Accuracy measure WRAcc [5]). However, such patterns cover generally a small number of objects. They are difficult to identify with most of the algorithms that exploit an exhaustive exploration of the search space with the help of a minimal frequency constraint (See, e.g., SD-MAP [6]). Therefore, *heuristic approaches* that are often based on *beam search* or *sampling* techniques are used to find only subsets of the interesting patterns. An

open problem concerns the computation of non-redundant patterns, avoiding to return thousands of variations for Boolean and/or numerical patterns [7]. Heuristic methods for sequential data have not yet attracted much attention. [8] introduced a sampling approach that is dedicated to the frequency measure only. A promising recent proposal is [9]: the sampling method `misère` can be used for any quality measure when exploiting sequences of events. Their key idea, is to draw patterns as strict generalizations of object descriptions while a time budget enables it, and to keep a pool of the best non redundant patterns found so far. Such a generic approach has been the starting point of our research though we were looking for further quality assessment of discovered subgroups.

Our contribution provides a search space exploration for labeled sequences of itemsets (and not just items) called `SeqScout`. It is based on sampling guided by a multi-armed bandit model followed by a generalization step, and a phase of local optimization. We show that it gives better results than a simple adaptation of `misère` with the same budget when considering huge search spaces. Our algorithm has several advantages: it gives results anytime and it benefits from random search to limit redundancy of results and to increase subgroup diversity. It is rather easy to implement and can be parallelized. It is agnostic w.r.t. the used quality measure and it finds local optima only (w.r.t. the considered quality measure).

Section II discusses related work. We formally define the problem in Section III. Section IV describes `SeqScout`. Section V presents an empirical study on several datasets, including applications on Starcraft II data (game analytics) and abstracts of articles published in the Journal of Machine Learning and Research.

II. Related Work

Sequential pattern mining [10] is a classical data mining task. It remains a challenging task due to the size of the search space. For instance, Raïssi and Pei [11] have shown that the number of sequences of length k is $w_k = \sum_{i=0}^{k-1} w_i \binom{|\mathcal{I}|}{k-i}$, with $|\mathcal{I}|$ being the number of possible items. As an example, if we consider the dataset **promoters** [12], $|\mathcal{I}| = 4$ and $k = 57$ (see Table II), the size of the search space is approximately 10^{41}. Various methods have been proposed to mine interesting sequential patterns. We review them briefly and we discuss their relevancy when considering our need for discriminative patterns.

A. Enumeration-based methods

Many *enumeration techniques* enable to mine patterns from Boolean, sequential and graph data [13]. They can be adapted for the case of discriminative pattern mining. For instance, the SPADE algorithm [14] has been adapted for sequence classification based on frequent patterns [15]. The main idea of such methods is to visit each candidate pattern only once while pruning large parts of the search space. Indeed, we know how to exploit formal properties (e.g., monotonicity) of many user-defined constraints (and not only the minimal frequency constraint). The quality measure is thus computed either during or after the discovery of all frequent patterns [16]. This is inefficient for the discovery of the best discriminative patterns only. To overcome this limitation and to help pruning the search space, upper bounds on the quality measure can be used. However, they remain generally too optimistic and are specific to a particular measure (See, e.g., [17]). Moreover, enumeration techniques coupled to upper bounds aim at solving the problem of finding the best pattern in the search space, and not the best pattern set.

B. Heuristic methods

An interesting current trend to support pattern discovery is to avoid exhaustive search and to provide anytime sets of high quality patterns, ideally with some guarantees on their quality. Examples of such guarantees are the distance to the best solution pattern [18] or the guarantee that the best solution can be found given a sufficient budget [7]. Let us discuss some of the heuristic approaches that have been proposed so far.

Beam search is a widely used heuristic algorithm. It traverses the search space level-wise from the most general to the most specific patterns and it restricts each level to a subset of non redundant patterns of high quality [19]. The greedy nature of beam search is its major drawback. Boley *et al*. proposed a two-step sampling approach giving the guarantee to sample patterns proportionally to different measures: frequency, squared frequency, area, or discriminativity [20]. However this method only works on those measures. Diop *et al.* proposed an approach which guarantees that the probability of sampling a sequential pattern is proportional to its frequency [8]. It focuses on the frequency measure only. Egho *et al.* have proposed the measure agnostic method `misère` [9]. Given a time budget, their idea is to generate random sequential patterns covering at least one object while keeping a pool of the best patterns obtained so far. It provides a result anytime, empirically improving over time.

Sequential pattern mining can be modeled as a multi-armed bandit problem enabling an *exploitation/exploration trade-off* [21]. Each candidate sequence is an "arm" of a bandit. For instance, Bosc *et al.* have developed such a game theory framework using Monte Carlo Tree Search to support subgroup discovery from labeled itemsets [7]. They proposed an approach based on sampling, where each draw improves the knowledge about the search space. Such a drawn object guides the search to achieve an exploitation/exploration trade-off.

To the best of our knowledge, the problem of mining discriminative sequences of itemsets with sampling approaches has not been addressed yet in the literature. We introduce our

SeqScout method that computes top-k non-redundant discriminative patterns, i.e., tackling a NP-hard problem [22]. For that purpose, we want hereafter to maximize the well known quality measure called $WRAcc$ [5]. Even though we focus on it, our method is generic enough for using any quality measure, without requiring specific properties. SeqScout does not require parameter tuning, unlike Beam Search. In contrast to misère, we have the guarantee to find local optima thanks to a hill-climbing search.

III. Non-redundant Subgroup Discovery: Preliminaries and Problem Statement

Let $\mathcal{I}$ be *a set of items*. Each subset $X \subseteq \mathcal{I}$ is called an *itemset*. A *sequence* $s = \langle X_1...X_n \rangle$ is an ordered list of $n > 0$ itemsets. The $size$ of a sequence s is denoted n, and $l = \sum_{i=1}^{n} |X_i|$ is its $length$. A database $\mathcal{D}$ is a set of $|\mathcal{D}|$ sequences (See an example in Table I). Given a set of classes $\mathcal{C}$, we denote by $\mathcal{D}_c \subseteq \mathcal{D}$ the set of sequences in $\mathcal{D}$ that are labeled by $c \in \mathcal{C}$.

Definition 1 (Subsequence)**.** A sequence $s = \langle X_1...X_{n_s} \rangle$ is a *subsequence* of a sequence $s' = \langle X'_1...X'_{n'_s} \rangle$, denoted $s \sqsubseteq s'$, iff there exists $1 \leq j_1 < ... < j_{n_s} \leq n'_s$ such that $X_1 \subseteq X'_{j_1}...X_{n_s} \subseteq X'_{j_{n_s}}$. In Table I, $\langle \{a\}\{b,c\} \rangle$ is a subsequence of s_1 and s_2.

Definition 2 (Extent, support and frequency)**.** The *extent* of a sequence s is $ext(s) = \{s' \in \mathcal{D} \mid s \sqsubseteq s'\}$. The *support* of a sequence s is $supp(s) = |ext(s)|$. Its *frequency* is $freq(s) = supp(s)/|\mathcal{D}|$. In Table I, $ext(\langle \{a\}\{b,c\} \rangle) = \{s_1, s_2\}$.

Definition 3 (set-extension)**.** A sequence s_b is a *set-extension* of $s_a = \langle X_1 X_2...X_n \rangle$ by x if $\exists i, 1 \leq i \leq n+1$ such that $s_b = \langle X_1...\{x\}_i...X_{n+1} \rangle$. In other words, we have inserted an itemset $X_i = \{x\}$ in the i^{th} position of s_a. $\langle \{a\}\{c\}\{b\} \rangle$ is a set-extension of $\langle \{a\}\{b\} \rangle$.

Definition 4 (item-extension)**.** A sequence s_b is an *item-extension* of $s_a = \langle X_1 X_2...X_n \rangle$ by x if $\exists i, 1 \leq i \leq n$ such that $s_b = \langle X_1...X_i \cup \{x\}, ..., X_{n+1} \rangle$. $\langle \{a,b\}\{b\} \rangle$ is an item-extension of $\langle \{a\}\{b\} \rangle$.

Definition 5 (Reduction)**.** A sequence s_b is a *reduction* of s_a if s_a is an set-extension or item-extension of s_b.

Definition 6 (Quality measure)**.** Let $\mathcal{S}$ be the set of all subsequences. A *quality measure* φ is an application $\varphi : \mathcal{S} \to \mathcal{R}$ that maps every sequence $s \in \mathcal{S}$ with a real number to reflect its interestingness (quality score). For instance, *Precision*, defined by $\mathcal{P}(s \to c) = \frac{supp(s,\mathcal{D}_c)}{supp(s,\mathcal{D})}$, is a quality measure.

Table I: An example database $\mathcal{D}$.

id	$s \in \mathcal{D}$	c
s_1	$\langle \{a\}\{a,b,c\}\{a,c\}\{d\}\{c,f\} \rangle$	$+$
s_2	$\langle \{a,d\}\{c\}\{b,c\}\{a,e\} \rangle$	$+$
s_3	$\langle \{e,f\}\{a,b\}\{d,f\}\{c\}\{b\} \rangle$	$-$
s_4	$\langle \{e\}\{g\}\{a,b,f\}\{c\}\{c\} \rangle$	$-$

Definition 7 (Local Optimum)**.** Let $N(s)$ be the *neighborhood* of s, i.e., the set of all item-extensions, set-extensions and reductions of s. $r^\star$ is a *local optimum* of $\mathcal{S}$ w.r.t. the quality measure φ iff $\forall r \in N(r^\star)$, $\varphi(r^\star) \geq \varphi(r)$.

Definition 8 (Non θ-redundant patterns)**.** A set of patterns, (also called subgroups or subsequences here), $\mathcal{S}_p \subseteq \mathcal{S}$ is *non θ-redundant* if given $\theta \in [0;1]$ and $\forall s1, s2 \in \mathcal{S}_p$, where $s_1 \neq s_2$, we have: $sim(s_1, s_2) \leq \theta$. In the following, we use the Jaccard index as a similarity measure:

$$sim(s_1, s_2) = \frac{|ext(s_1) \cap ext(s_2)|}{|ext(s_1) \cup ext(s_2)|}.$$

Problem Statement (Non-redundant subgroup set discovery)**.** For a database $\mathcal{D}$, an integer k, a real number θ, a similarity measure sim, and a target class $c \in \mathcal{C}$, the non redundant subgroup discovery task consists in computing the set $\mathcal{S}_p$ of the best non θ-redundant patterns of size $|\mathcal{S}_p| \leq k$, mined w.r.t the quality measure φ.

IV. Method Description

SeqScout is a sampling approach that exploits generalizations of database sequences, and search for local optima w.r.t. the chosen quality measure. Fig. 1 provides a global illustration for the method[1].

A. General overview

The main idea of the SeqScout approach is to consider each sequence of the labeled data as an arm of a multi-armed bandit when selecting the sequences for further generalization using the Upper Confidence Bound (UCB) principle (See Algorithm 1). The idea of the UCB is to give a score to each sequence that quantifies an exploitation/exploration trade-off, and to choose the sequence having the best one.

First (Lines 2-4), priority queues, π and $scores$, are created. π stores encountered patterns with their quality, and $scores$ keeps in memory the list of UCB scores of each sequence of the dataset, computed by using Equation 1 (See Section IV-B). $data_+$ contains the list of all sequences of the data set labeled with the target class. Indeed, taking sequences having the target class will lead to generalizations having at least one positive element. Then, the main procedure is launched as long as some computational budget is available. The best sequence w.r.t UCB is chosen (Line 9). This sequence is '*played*' (Line 10), meaning that it is generalized (See Section IV-C) and its quality is computed (See Section IV-G). The created pattern is added to π

[1]In the context of sequential pattern mining, the search space is a priori infinite. However, we can define the border of the search space (the bottom border in Fig. 1) by excluding patterns having a null support. We can easily prove that each element of this border is a sequence within the database. Therefore, the search space shape depends on the data.

(Line 11). Finally, the UCB score is updated (Line 12). As post processing steps, the top-k best non-redundant patterns are extracted from $scores$ using the filtering step (See Section IV-D). Finally, these patterns are processed by a local optimization procedure (See Section IV-E). Moreover, `SeqScout` needs other modules that concern the selection of the quality measure (See Section IV-F) and the quality score computation (See Section IV-G).

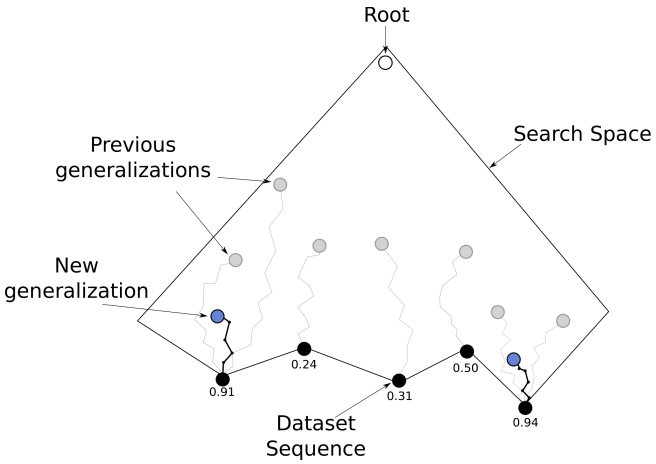

Figure 1: Illustration of `SeqScout`.

Algorithm 1 SeqScout

1: **function** SEQSCOUT($budget$)
2: $\pi \leftarrow PriorityQueue()$
3: $scores \leftarrow PriorityQueue()$
4: $data_+ \leftarrow FilterData()$
5: **for all** $sequence$ in $data_+$ **do**
6: $scores_{ucb}.add(sequence, \infty)$
7: **end for**
8: **while** budget **do**
9: $seq, qual, N_i \leftarrow scores.bestUCB()$
10: $seq_p, qual_p \leftarrow PlayArm(seq)$
11: $\pi.add(seq_p, qual_p)$
12: $scores.update(seq, \frac{N_i * qual + qual_p}{N_i+1}, N_i + 1)$
13: **end while**
14: $\pi.add(Optimize(\pi))$
15: **return** $\pi.topKNonRedundant()$
16: **end function**
17:
18: **function** OPTIMIZE(π)
19: $topK \leftarrow \pi.topKNonRedundant()$
20: **for all** $pattern$ in $topK$ **do**
21: **while** $pattern$ is not a local optima **do**
22: $pattern, qual \leftarrow BestNeighbor()$
23: **end while**
24: **end for**
25: **return** $pattern, qual$
26: **end function**

B. Sequence selection

We propose to model each sequence of the dataset as an arm of a *multi-armed bandit slot machine*. The action of playing an arm corresponds to *generalizing* this sequence to obtain a pattern, and the reward then corresponds to the *quality* of this pattern. Following an exploitation/exploration trade-off, sequences leading to bad quality patterns will be avoided, leading to the discovery of better ones.

The problem of the multi-armed bandit is well known in the Game Theory literature [23]. We consider a multi-armed bandit slot machine with k arms, each arm having its own reward distribution. Our problem is then formulated as follows. Having a number N of plays, what is the best strategy to maximize the reward? The more an arm is played, the more information about its reward distribution we get. However, to what extent is it needed to exploit a promising arm, instead of trying others that could be more interesting in the long term (explore)? Auer *et al.* proposed a strategy called *UCB1* [21]. The idea is to give each arm a score, and to choose the one that maximizes it:

$$UCB1(i) = \bar{x_i} + \sqrt{\frac{2ln(N)}{N_i}}, \quad (1)$$

where $\bar{x_i}$ is the empirical mean of the i^{th} arm, N_i is the number of plays of the i^{th} arm and N is the total number of plays. The first term encourages the exploitation of arms with good reward, while the second encourages the exploration of less played arms by giving less credit to the ones that have been frequently played.

Performing an exploitation/exploration trade-off for pattern mining has already been applied successfully to itemsets and numerical vectors by means of Monte Carlo Tree Search [7]. When dealing with a huge search space, using sampling guided by such a trade-off can give good results. However, contrary to [7], we consider the search space of extents, not intents. Exploring the extent search space guarantees to find patterns with non null support while exploring the intent search space leads towards multiple patterns with a null support. This is a crucial issue when dealing with sequences.

C. Pattern generalization

After the best sequence w.r.t. UCB1 is chosen, it is generalized, meaning that a new more general pattern is built. It enables to build a pattern with at least one positive element. Indeed, most of the patterns in the search space have a null support [11]. `SeqScout` generalizes a sequence s in the following way. It iterates through each item within each itemset $X_i \in s$, and it removes it randomly according

to the following rule:

$$\begin{cases} remain, & \text{if } z < 0.5 \\ remove, & \text{if } z \geq 0.5 \end{cases}, \text{ where } z \sim \mathcal{U}(0,1).$$

D. Filtering Step

To limit the redundancy of found patterns, a filtering process is needed. We adopt a well-described set covering principle from the literature (See, e.g., [7], [24]), summarized as follows. First, we take the best element, and then we remove those that are similar within our priority queue π. Then, we take the second best, and continue this procedure until the k best non-redundant elements are extracted.

E. Local optimum search

Finally, a local optimum search is launched w.r.t. Definition 7. Various strategies can be used. The first possible strategy is the Steepest Ascend Hill Climbing. It computes the neighborhood of the generalized pattern, i.e., all its item-extensions, set-extensions and reductions. Then, it selects the pattern among those of the neighborhood maximizing the quality measure. This is repeated until there is no more patterns in the neighborhood having a better quality measure. Another possible strategy is the Stochastic Hill Climbing: a neighbor is selected at random if its difference with the current one is large "enough". Notice however that it introduces a new parameter. Depending on the dataset, the branching factor can be very important. Indeed, for m items and n itemsets in the sequence, there are $m(2n+1)$ patterns in its neighborhood (See Theorem 1). To tackle this issue, we use First-Choice Hill Climbing [25]. We compute the neighborhood until a better pattern is created, then we directly select it without enumerating all neighbors.

Theorem 1. *For a sequence s, let n be its size, l its length, and m the number of possible items, the number of neighbors of s, denoted $|N(s)|$, is $m(2n+1)$.*

Proof: The number of item-extensions is given by:

$$|Iext| = \sum_{i=1}^{n} |\mathcal{I}| - |X_i| = nm - \sum_{i=1}^{n} |X_i| = nm - l.$$

We have now to sum the number of reductions, set-extensions and item-extensions:

$$|N(s)| = l + m(n+1) + |Iext| = m(2n+1).$$

■

F. Quality measure selection

The choice of the quality measure φ is application dependent. Our approach can deal with any known measures that support class characterization, such as, e.g., the $F1$ score, informedness or the Weighted Relative Accuracy [5]. The later, the $WRAcc$, is commonly used for discriminant pattern mining and subgroup discovery. It compares the proportion of positive elements (i.e., sequences labeled with the target class) to the proportion of positive elements in the whole database. Let $c \in C$ be a class value and s be a sequence:

$$WRAcc(s,c) = freq(s) \times \left(\frac{supp(s, \mathcal{D}_c)}{supp(s, \mathcal{D})} - \frac{|\mathcal{D}_c|}{|\mathcal{D}|} \right).$$

It is a weighted difference between the precisions $\mathcal{P}(s \to c)$ and $\mathcal{P}(\langle\rangle \to c)$. The weight is defined as $freq(s)$ to avoid the extraction of infrequent subgroups. Indeed, finding very specific subgroups covering one positive element would result in a perfect quality value but a useless pattern. $WRAcc$ value ranges in [-0.25, 0.25] in the case of a perfect balanced data set, i.e., containing 50% of positive elements.

We consider objective quality measures that are based on pattern support in databases (whole dataset, or restricted to a class). Using random draws makes it particularly difficult as each draw is independent: we cannot benefit from the data structures that are so efficient for classical exhaustive pattern mining algorithms do (See, e.g., [26]).

G. Efficient computation of quality scores

To improve the time efficiency of support computing, bitset representations have been proposed. For instance, SPAM [26] uses a bitset representation of a pattern when computing an item- or set-extension at the end of a sequence. In our case, we consider that an element can be inserted anywhere. Therefore, we propose a bitset representation that is independent from the insertion position. Its main idea lies in keeping all bitset representations of encountered itemsets in a hash table (memoization), and then combining them to create the representation of the desired sequence. The main idea is given in Fig. 2. Assume we are looking for the bitset representation of $\langle\{ab\},\{c\}\rangle$. Let $\langle\{c\}\rangle$ be an already encountered pattern (i.e., its representation is known) while $\langle\{ab\}\rangle$ was not. This can not be handled by the SPAM technique as a new element has to be added *before* a known sequence. Our algorithm will first try to find the bitset representation of $\langle\{ab\}\rangle$. As it does not exist yet, it will be generated and added to the memoization structure. Then, position options for the next itemset are computed (Line 2 in Fig. 2). The latter is then combined with a bitset representation of $\langle\{c\}\rangle$ using bitwise AND (Line 4). The support of the generated sequence can then be computed.

V. Experiments

We report an extensive empirical evaluation of `SeqScout` using multiple datasets. All the experiments were performed on a machine equipped with Intel Core i7-8750H CPU with 16GB RAM. Algorithms mentioned hereafter are implemented in Python 3.6.

Datasets: We perform the evaluation using some benchmark datasets that are well known in the pattern mining community, namely **aslbu** [27], **promoters** [12], **blocks** [12], **context** [12], **splice** [12], and **skating** [28].

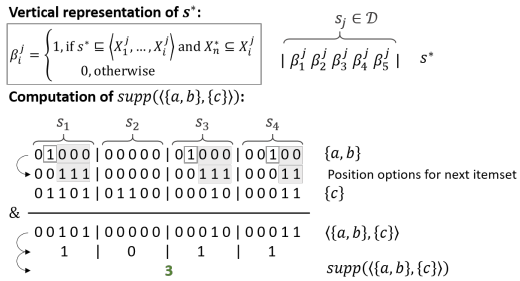

Figure 2: Bitset representation and support computing.

We also apply our algorithm on the real life dataset **sc2** that has been used in [16]. It was extracted from Starcraft IIa famous Real Time Strategy (RTS) game. The goal of each Starcraft match is to destroy units and buildings of the opponent. Three different factions exist, each with its own features (i.e., combat units, buildings, strategies). Roughly speaking, there is a duality between economy and military forces. Investing in military forces is important to defend or attack an opponent, while building economy is important to have more resources to invest into military forces. Sequences in **sc2** correspond to the buildings constructed during a game, and the class corresponds to the winner's faction. Once the target class (i.e., the winning faction) has been chosen, our algorithm will look for patterns of construction that characterize the victory of this faction.

Moreover, we use **jmlr**, a dataset consisting of abstracts of articles published in the Journal of Machine Learning Research [29]. Table II summarizes the statistics of datasets.

Baselines: We are not aware of available algorithms that address the problem of discriminative pattern mining in sequences of itemsets but several exist for processing sequences of events. Therefore, we compare `SeqScout` to two algorithms that we have modified to tackle sequences of itemsets, namely `misère` [9] and `BeamSearch` [24]. First, we implemented a simple extension of `misère` [9]. Second, we implemented a sequence-oriented version of a beam-search algorithm, denoted `BeamSearch`. To deal with sequences of itemsets, we consider item-extensions and set-extensions at each given depth. Moreover, for the sake of non-redundancy in the returned patterns, we modify its best-first search nature so that the expanded nodes get diverse, as defined in [24]. Indeed, without this modification, `BeamSearch` selects directly the best patterns at each level, that are often redundant. It may lead to a situation where almost all patterns are removed in the non-redundancy post-processing step, giving rise to solutions with less than k elements.

Settings: If not stated otherwise, we use the following settings. Each algorithm has been launched 5 times, and the reported results are averaged over these runs. For `BeamSearch`, we empirically set the parameter $width = 50$. For all algorithms, we set $\theta = 0.5$, $time\_budget = \infty$, $iteration\_num = 10,000$, and $top\_k = 5$. Note that instead of giving a fixed time budget for running an algorithm on each dataset, we chose to limit the number of iterations $iteration\_num$, one iteration corresponding to a single computation of the quality measure. Indeed, this computation is the most time consuming one: it needs for computing the extent w.r.t. the whole dataset. Therefore, using the same time budget on different datasets would not provide a fair comparison: having 50,000 iterations on a small dataset versus 50 on a complex one with the same time budget is not interesting.

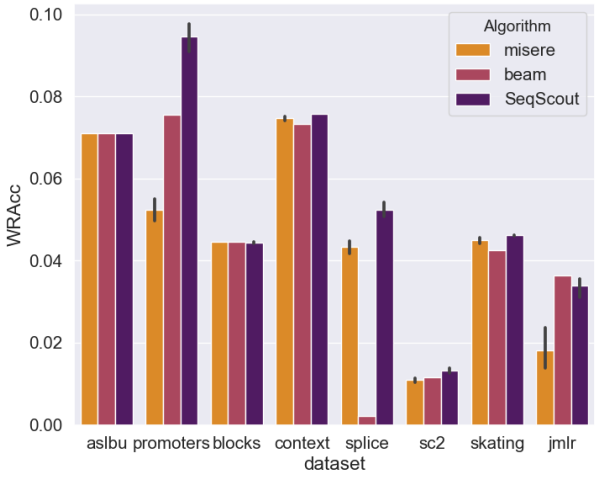

Figure 3: $WRAcc$ of top-5 best patterns (10K iterations)

A. *Performance evaluation using $WRAcc$*

To assess the performance of the algorithms, let us first use the mean of the $WRAcc$ of the top-k non redundant patterns given by algorithms `misère`, `BeamSearch` and `SeqScout`. Fig. 3 provides the comparative results. Our approach gives better results on the majority of datasets. Moreover, it remains quite stable, contrarily to `BeamSearch` which can be sometimes fairly inefficient (See on **splice**).

Table II: Datasets

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	l_{max}	Search Space Size
aslbu [27]	441	124	27	$4.45 * 10^{60}$
promoters [12]	106	4	57	$1.58 * 10^{41}$
blocks [12]	210	8	12	$2.74 * 10^{12}$
context [12]	240	47	123	$5.24 * 10^{224}$
splice [12]	3,190	8	60	$3.28 * 10^{62}$
sc2 [16]	5,000	30	30	$6.48 * 10^{48}$
skating [28]	530	41	120	$1.15 * 10^{212}$
jmlr [29]	788	3,836	228	$1.84 * 10^{853}$

Table III: Mean values of measures for top-5 patterns.

Dataset	Informedness			F1		
	misere	BeamS	SeqScout	misere	BeamS	SeqScout
aslbu	0.195	**0.198**	**0.198**	0.505	0.505	0.505
promoters	0.039	0.089	**0.113**	0.512	0.545	**0.580**
blocks	**0.399**	0.394	0.393	0.381	0.370	**0.382**
context	0.436	0.437	**0.469**	0.528	0.532	**0.584**
splice	0.342	0.006	**0.356**	0.394	0.082	**0.471**
sc2	0.004	0.004	**0.010**	0.524	0.519	**0.526**
skating	0.396	0.385	**0.415**	0.361	0.380	**0.389**
jmlr	0.224	**0.403**	0.355	0.143	0.236	**0.243**

Table IV: Number of found local optima - 10s budget

Dataset	Integer Set	Bitset	Variation(%)
aslbu	16,453	17,330	5
promoters	7,185	8,858	24
blocks	30,725	45,289	51
context	4,651	2,667	-47
splice	289	254	-12
sc2	943	605	-36
skating	4,001	1,283	-68
jmlr	704	31	-95

B. Quality with respect to the number of iterations

We discuss the result quality in terms of $WRAcc$ over the number of iterations. Fig. 4, 5, 6 depict the results for the top-5 non-redundant patterns on datasets **promoters**, **skating** and **sc2**. Note that for the same data, the results may vary from run to run, due to the random component of `misère` and `SeqScout`. It explains some fluctuations of the quality. Nevertheless, for each $iteration\_num$ setting, `SeqScout` has shown better results.

C. Using other quality measures

To empirically illustrate the measure agnostic characteristic of `SeqScout`, we have used other quality measures: $F1$-score and $Informedness$. The results are shown in Table III. Our algorithm gives generally better results.

D. Performance study under varying θ

We evaluate the performance of the algorithms when varying the similarity threshold θ. Fig. 7 shows the performance on the dataset **promoters**. We did not included the results for $\theta = 0$ because it would mean finding patterns with totally disjoint extents. It results in finding a number of patterns lesser than k for all algorithms, such that the mean would be misleading. We can see from the plot that `SeqScout` outperforms other algorithms for all θ values.

E. Performance study under varying k (top-k)

We investigate the performance of the search for top-k patterns when changing the k parameter. Fig. 8 shows the results when considering the **context** dataset. `SeqScout` gives better results. Note that the mean $WRAcc$ decreases for all algorithms, as increasing k leads to the selection of lower quality patterns.

F. Quality versus search space size

We evaluate the average $WRAcc$ of the top-5 patterns w.r.t. the maximum length of sequences (See Fig. 9). To do so, we have truncated the dataset to control the maximum lengths of sequences. We demonstrate it on the **skating** dataset. The plot shows that `SeqScout` gives better average $WRAcc$ values whatever the search space size is. We also note a general decrease of quality when increasing the search space size. Indeed, some patterns that are good for a smaller maximum length can appear in negative elements for larger maximum lengths, resulting in a decreasing quality of patterns. Note that the opposite phenomenon can also appear.

G. Comparison to ground truth

We now compute the ratio of the $WRAcc$ result quality of `SeqScout` to the one of the ground truth. Therefore, we have launched an exhaustive algorithm (Breadth-First-Search) on the dataset **sc2**. However, the search space size makes the problem intractable in a reasonable time. To tackle this issue, we truncate our dataset to have sequences with a length of 10. Note that even with this severe simplification, our exhaustive algorithm took 35 hours to conclude, and found more than 40 billions of patterns. As we can see in Fig. 10, `SeqScout` improves over time, and it reaches more than 90% of the best pattern set in less than 2,000 iterations (35 seconds on our machine).

H. Sequence length

The pattern lengths on all datasets are reported in Fig. 11. Let us consider the **splice** dataset: `BeamSearch` gives short patterns (max 8), which is significantly less than `SeqScout` and `misère`. This may explain why the `BeamSearch` result quality is bad on this dataset (See Fig. 3). One hypothesis could be that it does not have enough iterations to go deep enough in the search space. Another hypothesis is that `BeamSearch` cannot find good patterns having bad parents w.r.t. $WRAcc$: its good patterns are short ones.

I. Bitset vs. Integer set representation

We investigate the usefulness of bitset representation by comparing it against an integer set representation where each item is represented by an integer. Therefore, we have compared the number of iterations `SeqScout` made for a fixed time budget on each dataset. We set $time\_budget = 10$ seconds. The results are summarized in Table IV. We can see that the bitset representation gives a performance gain for datasets with smaller search space size upper bound. Indeed, **context**, **skating**, and **jmlr** have sequences of large lengths, leading to large bitset representations. Those bitsets are then split into different parts to be processed by the CPU. If the number of splits is too large, the bitset representation becomes inefficient, and using a classical integer set representation is a better option.

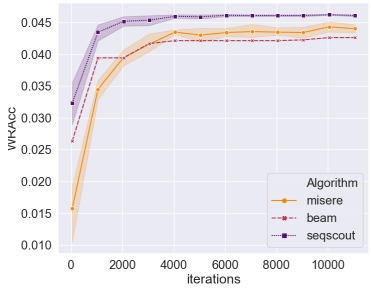

Figure 4: $WRAcc$ for top-5 patterns w.r.t. iterations (**skating**)

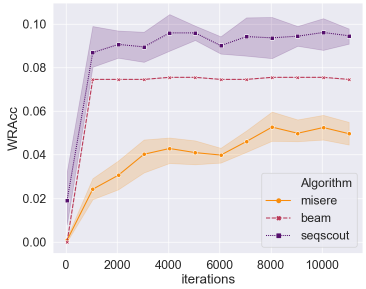

Figure 5: $WRAcc$ for top-5 patterns w.r.t. iterations (**promoters**)

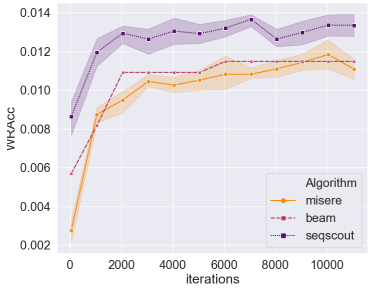

Figure 6: $WRAcc$ for top-5 patterns w.r.t. iterations (**sc2**)

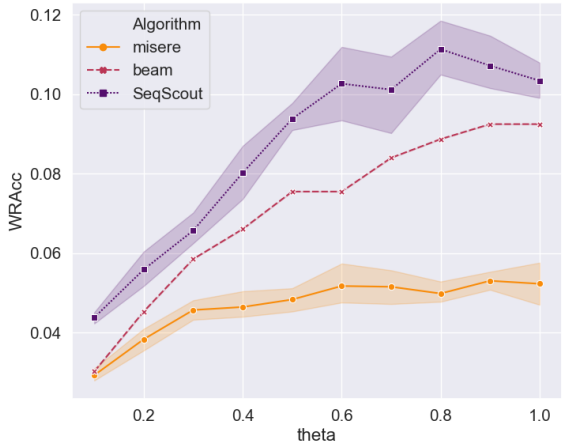

Figure 7: $WRAcc$ of top-5 patterns versus similarity threshold θ (**promoters**)

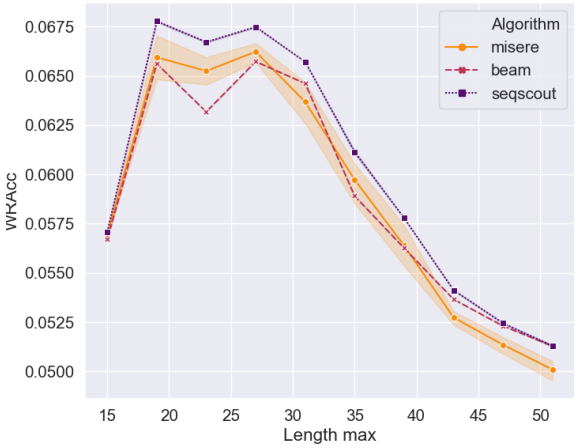

Figure 9: $WRAcc$ of top-5 patterns versus max length on (**skating**)

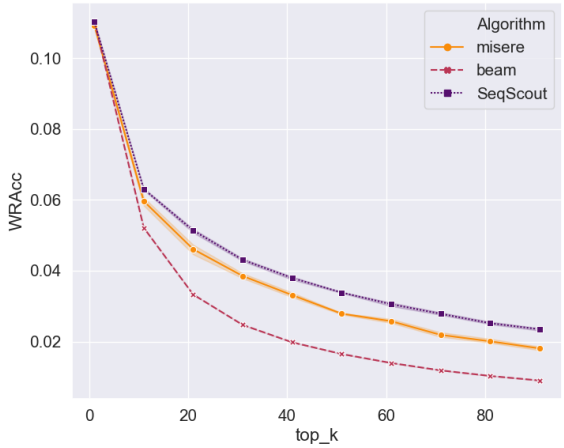

Figure 8: $WRAcc$ of top-k patterns vs. k (**context**)

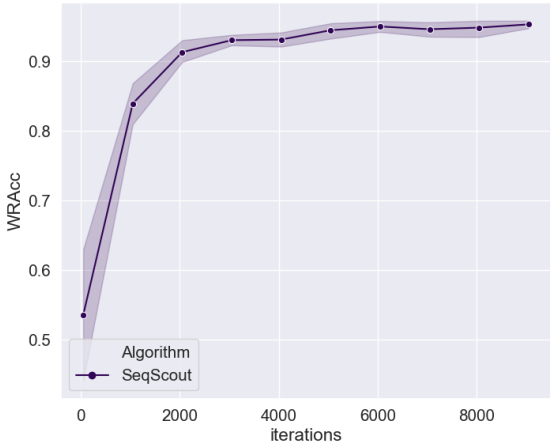

Figure 10: Ratio of $WRAcc$ of `SeqScout` to ground truth (**sc2**)

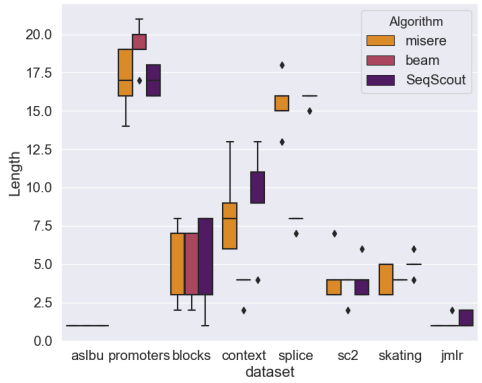

Figure 11: Length of top-5 best patterns - 10K iterations

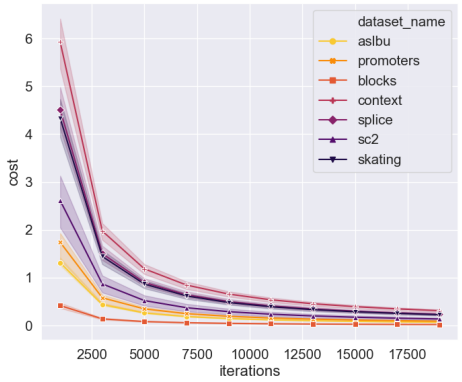

Figure 12: Additional cost of local optima search

J. Number of iterations during the local optima search

During the local optima search step, an additional number of iterations is made to improve the qualities of the best patterns. In Fig. 12, we plot the ratio of the additional iterations necessary for local optima search w.r.t. the number of iterations given in the main search. The more iterations we have, the more negligible the ratio is. Based on that analysis, we may infer that a rather fair number of iterations for our experiments is 10,000. However, note that we did not plot the additional cost of **jmlr**. Indeed, in the particular case of text data, the number of possible items is quite large, leading to a fairly long local optima search. Consequently, we note that `SeqScout` may not be the relevant choice with this kind of dataset.

K. Quality evaluation on **sc2**

Given a faction, what are the best strategies to win? `SeqScout` will look for patterns of construction which are characteristic to the victory of this faction. Let us consider the "Terran" faction, as an example. For this faction, one of the best pattern found is: $\langle\{Hatchery\}, \{Supply\}, \{Factory\}, \{Supply\}\rangle$. We use the colors as indicators of fractions: blue for "Terran" and purple for "Zerg". We can see that the "Terran" player is investing in military units ($\{Supply\}$ and $\{Factory\}$). The "Zerg" player chooses to invest in its economy by creating a $\{Hatchery\}$: she fosters the so-called late game, sacrificing its military forces in the so-called early game. In such a scenario, the "Terran" strikes in early game, knowing its military advantage, and she tends to win the match. This example shows that our algorithm can provide relevant patterns that may be useful, e.g., to identify unbalanced strategies [16].

L. Quality evaluation on **jmlr**

Let us now use a dataset consisting of abstracts of articles from JMLR journal [29]. Sentences have been tokenized and stemmed. Each sequence is then a sequence of items, where an item corresponds to a stemmed word. To label sequences, we chose to add a class '+' to sequences containing a target word, and a class '-' to others. We also removed the target word from the sequences, as it would lead to the discovery of patterns having the target word, which is not interesting. The goal is then to find sequences of items (words), characteristic for the target word. The results for two target words (*svm* and *classif*) are presented in Table V. As we can see, these patterns are indeed characteristics of the target words.

Table V: Top-10 non-singleton patterns given by `SeqScout` on **jmlr** for target words: "*svm*" and "*classif*"

svm	classif
effici method	classifi set
requir support	classifi value
propos paper	gener real
formul classif	problem condit
individu show	learn error
scale vector	support gener
base number support	paper propos
support machin present show	problem classification set
machin techniqu	problem appli
solut number	data gener

VI. Conclusion

We presented the algorithm `SeqScout` to discover relevant subgroups in sequences of itemsets. Though we are not aware of available algorithms to solve the same problem, we have implemented adaptations of two other algorithms when considering sequences of itemsets, namely `misère` and `BeamSearch`. Our experiments showed that `SeqScout` gave better results, without the need for additional parameter tuning as in the case of `Beam Search`. However we can note that due to the nature of the bandit approach, this algorithm perfoms better on dataset of reasonnable size. Parallelizing the algorithm with an efficient programming language could be interesting to tackle this issue in future works. We plan to further improve our method

by incorporating the Monte Carlo Tree Search, a logical evolution of bandit based method to sample the search space. Indeed, instead of repeatedly generalizing promising dataset sequences, it could be more interesting to generalize promising patterns that occur in such promising sequences, identifying the best parts of the search space as the number of iterations increases.

Acknowledgment This research has been partially funded by the French National project FUI DUF 4.0 2017-2021.

REFERENCES

[1] P. K. Novak, N. Lavrač, and G. I. Webb, "Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining," *Journal Machine Learning Research*, vol. 10, pp. 377–403, Jun. 2009.

[2] S. Wrobel, "An algorithm for multi-relational discovery of subgroups," in *Proceedings PKDD 1997*, pp. 78–87.

[3] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *SIGKDD Explor. Newsl.*, vol. 12, no. 1, pp. 40–48, Nov. 2010.

[4] B. Letham, C. Rudin, and D. Madigan, "Sequential event prediction," *Machine Learning*, vol. 93, no. 2, pp. 357–380, Nov 2013.

[5] N. Lavrac, P. A. Flach, and B. Zupan, "Rule evaluation measures: A unifying view," in *Proceedings ILP 1999*, pp. 174–185.

[6] M. Atzmller and F. Puppe, "SD-Map – a fast algorithm for exhaustive subgroup discovery," in *Proceedings PKDD 2006*, pp. 6–17.

[7] G. Bosc, J.-F. Boulicaut, C. Raïssi, and M. Kaytoue, "Anytime discovery of a diverse set of patterns with monte carlo tree search," *Data Min. Knowl. Discov.*, vol. 32, no. 3, pp. 604–650, May 2018.

[8] L. Diop, C. T. Diop, A. Giacometti, D. Li Haoyuan, and A. Soulet, "Sequential Pattern Sampling with Norm Constraints," in *Proceedings IEEE ICDM 2018*, pp. 89–98.

[9] E. Egho, D. Gay, M. Boullé, N. Voisine, and F. Clérot, "A user parameter-free approach for mining robust sequential classification rules," *Knowl. Inf. Syst.*, vol. 52, no. 1, pp. 53–81, Jul 2017.

[10] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings IEEE ICDE 1995*, pp. 3–14.

[11] C. Raïssi and J. Pei, "Towards bounding sequential patterns," in *Proceedings ACM SIGKDD 2011*, pp. 1379–1387.

[12] D. Dua and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[13] A. Giacometti, D. H. Li, P. Marcel, and A. Soulet, "20 years of pattern mining: a bibliometric survey," *SIGKDD Explor. Newsl.*, vol. 15, no. 1, pp. 41–50, 2013.

[14] M. J. Zaki, "Spade: An efficient algorithm for mining frequent sequences," *Machine Learning*, vol. 42, no. 1, pp. 31–60, Jan 2001.

[15] C. Zhou, B. Cule, and B. Goethals, "Pattern based sequence classification," *IEEE Trans. Knowl. Data Eng.*, vol. 28, pp. 1285–1298, 2016.

[16] G. Bosc, P. Tan, J.-F. Boulicaut, C. Raïssi, and M. Kaytoue, "A Pattern Mining Approach to Study Strategy Balance in RTS Games," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 9, no. 2, pp. 123–132, Jun. 2017.

[17] S. Nowozin, G. Bakir, and K. Tsuda, "Discriminative subsequence mining for action classification," in *Proceedings IEEE ICSV 2007*, Oct, pp. 1–8.

[18] A. Belfodil, A. Belfodil, and M. Kaytoue, "Anytime subgroup discovery in numerical domains with guarantees," in *Proceedings ECML/PKDD 2018 Part 2*, pp. 500–516.

[19] W. Duivesteijn, A. J. Feelders, and A. Knobbe, "Exceptional model mining," *Data Min. Knowl. Discov.*, vol. 30, no. 1, pp. 47–98, Jan 2016.

[20] M. Boley, C. Lucchese, D. Paurat, and T. Gärtner, "Direct local pattern sampling by efficient two-step random procedures," in *Proceedings ACM SIGKDD 2011*, pp. 582–590.

[21] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, May 2002.

[22] L. Qin, J. X. Yu, and L. Chang, "Diversifying top-k results," *Proceedings VLDB Endow.*, vol. 5, no. 11, pp. 1124–1135, Jul. 2012.

[23] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *CoRR*, vol. abs/1204.5721, 2012.

[24] M. van Leeuwen and A. J. Knobbe, "Diverse subgroup set discovery," *Data Min. Knowl. Discov.*, vol. 25, no. 2, pp. 208–242, 2012.

[25] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[26] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu, "Sequential pattern mining using a bitmap representation," in *Proceedings ACM SIGKDD 2002*, pp. 429–435.

[27] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos, "Discovering frequent arrangements of temporal intervals," in *Proceedings IEEE ICDM 2005*, pp. 354–361.

[28] F. Mörchen and A. Ultsch, "Efficient mining of understandable patterns from multivariate interval time series," *Data Min. Knowl. Discov.*, vol. 15, no. 2, pp. 181–215, Oct 2007.

[29] N. Tatti and J. Vreeken, "The long and the short of it: Summarising event sequences with serial episodes," *CoRR*, vol. abs/1902.02834, 2019.