HAL
open science

Anytime mining of sequential discriminative patterns in labeled sequences

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

To cite this version:

Romain Mathonat, Diana Nurbakova, Jean-François Boulicaut, Mehdi Kaytoue. Anytime mining of sequential discriminative patterns in labeled sequences. Knowledge and Information Systems (KAIS), 2020, pp. 439-476. 10.1007/s10115-020-01523-7 . hal-03000696

HAL Id: hal-03000696

https://hal.science/hal-03000696v1

Submitted on 12 Nov 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Noname manuscript No.
(will be inserted by the editor)

Anytime Mining of Sequential Discriminative Patterns in Labeled Sequences

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

Received: 05 Nov 2019/ Revised: 30 Jun 2020/ Accepted: 11 Oct 2020

Abstract It is extremely useful to exploit labeled datasets not only to learn models and perform predictive analytics but also to improve our understanding of a domain and its available targeted classes. The *subgroup discovery task* has been considered for more than two decades. It concerns the discovery of patterns covering sets of objects having interesting properties, e.g., they characterize or discriminate a given target class. Though many subgroup discovery algorithms have been proposed for both transactional and numerical data, discovering subgroups within labeled sequential data has been much less studied. First, we propose an anytime algorithm `SeqScout` that discovers interesting subgroups w.r.t. a chosen quality measure. This is a sampling algorithm that mines discriminant sequential patterns using a multi-armed bandit model. For a given budget, it finds a collection of local optima in the search space of descriptions and thus, subgroups. It requires a light configuration and is independent from the quality measure used for pattern scoring. We also introduce a second anytime algorithm `MCTSExtent` that pushes further the idea of a better trade-off between exploration and exploitation of a sampling strategy over the search space. To the best of our knowledge, this is the first time that the Monte Carlo Tree Search framework is exploited in a

Romain Mathonat
Univ de Lyon, CNRS, INSA Lyon, LIRIS, UMR5205, F-69621, France
Atos, F-69100 Villeurbanne, France
romain.mathonat@insa-lyon.fr

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

Jean-François Boulicaut
Univ de Lyon, CNRS, INSA Lyon, LIRIS, UMR5205, F-69621, France
jean-francois.boulicaut@insa-lyon.fr

Mehdi Kaytoue
Univ de Lyon, CNRS, INSA Lyon, LIRIS, UMR5205, F-69621, France
Infologic R&D, F-69007 Lyon, France
mka@infologic.fr

sequential data mining setting. We have conducted a thorough and comprehensive evaluation of our algorithms on several datasets to illustrate their added-value, and we discuss their qualitative and quantitative results.

Keywords Pattern Mining, Subgroup Discovery, Upper Confidence Bound, Monte Carlo Tree Search , Multi-Armed Bandit

1 Introduction

In many data science projects, we have to process labeled data and it is often valuable to discover descriptions, say patterns, that discriminate well some classes. This can be used to support various machine learning techniques aiming at predicting the class label for unseen objects (i.e., learning models). It is also interesting *per se* since the language for descriptions is, by design, readable and interpretable by analysts and data owners. Therefore, it can be used to explore and better understand a given phenomenon thanks to collected labeled data and it can also provide a solid foundation for building new relevant features [43]. The search for such patterns has been referred to under different names, among which *subgroup discovery*, *emerging pattern mining* or *contrast set mining* [30]. Hereafter, we will use the terminology of the subgroup discovery framework [38].

Labeled sequential data are ubiquitous. This makes subgroup discovery applicable to many application domains, for instance, text or video data analysis [31], industrial process supervision [40], biomedical genomics sequential data analysis [35], web usage mining [32], video game analytics [8], etc. Let us consider a maintenance scenario for a cloud environment. Data generated by such a system are sequences of events, and can be labeled with failure, i.e., breakdown presence or absence. Applying classification techniques helps answering to: "*will* a failure event occur?" (see, e.g., [39]), while applying sequential event prediction helps determining "*what* is the next event to occur?" (see, e.g., [25]). 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. Considering again our cloud environment example, the goal is to compute patterns "that tend to occur with breakdowns". Such patterns provide valuable hypotheses for a better understanding of the system. Once validated by domain experts, the discovered patterns can then be used to support maintenance planning tasks.

Subgroup discovery on labeled sequential data faces several challenges. 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, e.g., the Weighted Relative Accuracy measure *WRAcc* [23]). However, such patterns cover generally a small number of objects. They are difficult to identify with most of the algorithms that perform an exhaustive exploration of the search space with the help of a minimal frequency constraint (see, e.g., SD-MAP [2] for categorical and numerical data only). Therefore, *heuristic approaches* that are often based on *beam search* or *sampling* techniques are

used to find only subsets of the interesting patterns. Another fairly 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. Diop et al. [13] introduced a sampling approach that draws patterns according to the frequency measure only. In [16], Egho et al., propose a promising method: 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 random generalizations of object descriptions while a time budget enables it, and to keep a pool of the best non-redundant patterns found so far. The strength of this algorithm is that it finds patterns covering at least one element. However, it does not exploit previous sampling to guide the search in the next iterations: it is an exploration-only framework, without memory. Such a generic approach has been the starting point of our research, though we were looking for further quality assessment of discovered subgroups.

Our research concerns search space exploration methods for labeled sequences of itemsets and not just sequences of items. We first describe the algorithm `SeqScout` that has been introduced in our conference paper [26]. `SeqScout` 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 an adaptation of `misère` for the case of sequences of itemsets with the same budget when considering huge search spaces. We then present the main contributions of this paper than can be summarized as follows:

- We introduce `MCTSExtent`, a significant evolution of `SeqScout` where the multi-armed bandit model evolves towards a Monte Carlo Tree Search (MCTS). Doing so, we look for a trade-off between exploration and exploitation during the search of interesting patterns. Using MCTS for pattern discovery has been proposed recently in [7] as a promising framework for pattern discovery in transactional and numerical data. Defining the right policies about the different MCTS operators remains however open even in such rather simple settings. To the best of our knowledge, our algorithm `MCTSExtent` is the first attempt to apply MCTS for pattern mining in sequential data.
- To the best of our knowledge, there is no solution for the problem of *the Longest Common Subsequence* for sequences of itemsets, that is needed within `MCTSExtent` when generalizing subsequences. Thus, we propose a new dynamic programming procedure to solve it.
- We provide a thorough assessment of our claims via an exhaustive set of experiments with benchmark data involving the comparison with the competitor algorithms, namely `misère`, `Beam Search` and `SeqScout`.
- We describe a novel application of the problem of mining sequential discriminative patterns in labeled sequences to the e-sport domain for which our algorithm helps discovering actionable play patterns. It exploits *Rocket League* video game data[1]. We make our original dataset publicly available for the purpose of a better evaluation and reproducibility.

Both presented algorithms have several advantages: they give results anytime and take benefits from random search to limit redundancy of results and to increase

[1] `https://www.rocketleague.com/`

subgroup diversity. They are also agnostic w.r.t. the used quality measure. All source codes, original datasets and experimental results are available online[2].

The paper is organised as follows. Section 2 discusses the related work. We formally define the problem in Section 3. We then describe our solution algorithms `SeqScout` in Section 4 and `MCTSExtent` in Section 5. Section 6 presents an empirical study on several datasets, including experiments with Starcraft II and Rocket League data (game analytics). Section 7 concludes.

2 Related Work

Sequential pattern mining is now a classical data mining task and was introduced by the pioneer contribution of Agrawal et al. [1], that tackles frequent sequential pattern mining. It remains a challenging task due to the size of the search space, that is, the set of all possible sequential pattern, bounded by the length of the biggest sequence of the dataset. Thus, Raïssi and Pei have shown in [33] 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 well-known UCI dataset **promoters** [14], with $|\mathcal{I}| = 4$ and $k = 57$, the size of the search space is approximately 10^{41}.

Various methods have been proposed to mine interesting sequential patterns within a constraint-based data mining approach. We review them briefly and we discuss their relevancy when considering our need for discriminative patterns.

2.1 Enumeration-based Methods

Many *enumeration techniques* enable to mine patterns from Boolean, numerical, sequential and graph data [19]. They can be adapted for the case of discriminative pattern mining. For instance, the SPADE algorithm [41] has been adapted for sequence classification based on frequent patterns [42]. 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). Their quality measure is thus computed either during or after the discovery of all frequent patterns [8]. This is inefficient for the discovery of the best discriminative patterns only. To overcome this limitation and to support 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., [31], [18]). Moreover, enumeration techniques coupled to upper bounds mainly aim at solving the problem of finding the best pattern in the search space and not the best pattern set. [22] also proposed an interesting approach based on search space pruning for finding the best discriminative subsequences under a gap constraint. However, their work is also specific to only one discriminative quality measure. It requires to tune several parameters, including a minimum support, and it has been designed for processing sequences of items, and not sequences of itemsets.

[2] `https://github.com/Romathonat/MCTSExtent`

2.2 Heuristic Methods

An interesting trend to support pattern discovery is to avoid exhaustive search and to provide high quality patterns available anytime during the search, ideally with some guarantees on their quality. Examples of such guarantees are the distance to the best solution pattern [5] 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 (often structured as a lattice) 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 [15]. The greedy and *exploitation-only* nature of beam search is its major drawback, yet it allows to quickly discover some interesting patterns.

Gsponer *et al.* used a different approach, where a linear model is trained on a set of features extracted from sequences of the dataset [20]. Those features correspond to the search space of all possible subsequences. By minimizing a loss function, they can directly look at weights of their model to determine the most predictive subsequences. However, this approach is only applied on sequences of items, with numeric classes, and it can not choose a quality measure to optimize. For example, using this algorithm to find patterns corresponding to a large subgroup, *i.e.*, more generalistic predictive rule covering many instances of the dataset with a lesser proportion of positive element, is not possible.

Boley *et al.* proposed a two-step sampling approach giving the guarantee to sample patterns (on itemsets) proportionally to different measures, namely: frequency, squared frequency, area, or discriminativity [6]. However this method only works on these measures. To consider another measure, Moens and Boley had to design a new method [28]. Considering sequences, Diop *et al.* proposed an approach which guarantees that the probability of sampling a sequential pattern is proportional to its frequency [12, 13]. It focuses on the frequency measure only.

Egho *et al.* have proposed the measure agnostic method `misère` [16]. 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, but there is no use of previous sampling: this is an *exploration-only* strategy.

Pattern mining can be modeled as a multi-armed bandit problem enabling an *exploitation/exploration trade-off* [3]. Each candidate sequence is an "arm" of a bandit. Bosc *et al.* have developed such a game theory framework using Monte Carlo Tree Search to support subgroup discovery from labeled categorical and numerical data [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 agnostic of the chosen quality measure with sampling approaches has not been addressed yet in the literature, except in our recent conference paper [26]. Hereafter, we describe our methods `SeqScout` and `MCTSExtent` that compute top-k non-redundant discriminative patterns. For that purpose, we want to maximize the well known quality measure called $WRAcc$ [23]. Even though we focus on it, our methods are generic enough for using any quality measure, without requiring specific properties. For instance, we report the results obtained for other measures

Table 1: Notations

Notation	Description
$\mathcal{I}$	set of possible items
$m = \|\mathcal{I}\|$	number of possible items
$x \in \mathcal{I}$	item
$X \subseteq \mathcal{I}$	itemset
$\mathcal{D}$	database
C	set of classes
$\mathcal{S}$	set of all subsequences, i.e., search space
$s = \langle X_1 ... X_n \rangle$	sequence of itemsets
X_i^j	the ith itemset in s_j
n	size of a sequence $s = \langle X_1 ... X_n \rangle$
$l = \sum_{i=1}^{n} \|X_i\|$	length of a sequence
$c \in C$	class
$s \sqsubseteq s'$	s is a subsequence of s'
$ext(s)$	extent of s
$supp(s)$	support of s
$freq(s)$	frequency of s
φ	quality measure
$Neighborhood(s)$	neighborhood of s

in Section 6.7. Note also that our algorithms do not require parameter tuning, unlike `Beam Search`.

3 Formalizing the Non-redundant Subgroup Discovery Task

Let us now formalize our pattern mining task. 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 as n, and $l = \sum_{i=1}^{n} |X_i|$ is its *length*. A database $\mathcal{D}$ is a set of $|\mathcal{D}|$ sequences (see Table 2). 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 $c \in \mathcal{C}$. We summarize the notations in Table 1.

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 2, $\langle \{a\}\{b,c\} \rangle$ is a subsequence of s_1 and s_2.

Definition 2 (Positive element/sequence/object) A sequence of the database labeled with the target class is called a *positive element*.

Table 2: 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 3 (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 $freq(s) = supp(s)/|\mathcal{D}|$. Given the data in Table 2, we have $ext(\langle\{a\}\{b,c\}\rangle) = \{s_1, s_2\}$.

Definition 4 (Set-extension) A sequence s_b is a *set-extension* by $x \in \mathcal{I}$ of a sequence $s_a = \langle X_1 X_2 ... X_n\rangle$ 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 ith position of s_a.

Definition 5 (Item-extension) A sequence s_b is an *item-extension* by $x \in \mathcal{I}$ of a sequence $s_a = \langle X_1 X_2 ... X_n\rangle$ if $\exists i, 1 \leq i \leq n$ such that $s_b = \langle X_1 ... X_i \cup \{x\}, ..., X_{n+1}\rangle$.

For example, $\langle\{a\}\{c\}\{b\}\rangle$ is a set-extension of $\langle\{a\}\{b\}\rangle$ and $\langle\{a,b\}\{b\}\rangle$ is an item-extension of $\langle\{a\}\{b\}\rangle$.

Definition 6 (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 7 (Quality measure) Let $\mathcal{S}$ be the set of all possible subsequences in a dataset. A *quality measure* φ is an application $\varphi : \mathcal{S} \rightarrow \mathcal{R}$ that maps every sequence from $s \in \mathcal{S}$ with a real number to reflect its interestingness (quality score in the data). For instance, *Precision*, defined by $\mathcal{P}(s \rightarrow c) = \frac{supp(s,\mathcal{D}_c)}{supp(s,\mathcal{D})}$, is a quality measure about the association of a class label c with a sequence s.

Definition 8 (Local optimum) Let $Neighborhood(s)$ 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 $\forall r \in Neighborhood(r^\star)$, $\varphi(r^\star) \geq \varphi(r)$.

Definition 9 (Non θ-redundant subsequences) A set of patterns $\mathcal{S}_p \subseteq \mathcal{S}$ 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: $sim(s_1, s_2) \leq \theta$, where sim is a similarity function. We use here the Jaccard index as a similarity measure as in [24]:

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

We can now precisely define the considered mining task.

Problem Statement: For a database $\mathcal{D}$, an integer k, a real number θ, a similarity measure sim, a quality measure φ 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.

This problem can be illustrated in a visual more visual way. A sequence of itemsets can be represented as shown in Fig. 1. Each vertical slice corresponds to an itemset, and each square of colour represents an item within this itemset. For instance, this sequence of itemsets could be written as follows: $s_1 = \langle\{greySquare\}, \{greySquare, brownSquare\}, \{greySquare\}, \{greySquare, blueSquare\}...\rangle$. A dataset of sequences of itemsets is then composed of different sequences like represented in Fig. 1 and Fig. 2. Our goal is then to extract patterns, i.e. subsequences, appearing in dataset, whose quality is assessed with

Fig. 1: Sequence of itemsets seq_1

Fig. 2: Sequence of itemsets seq_2

Fig. 3: A pattern p appearing in sequence of itemsets s_1. The items not belonging to the pattern are given in light grey.

Fig. 4: A pattern p appearing in sequence of itemsets s_2. The items not belonging to the pattern are given in light grey.

quality measures, like the $WRAcc$. The pattern p (given in colour) appears in both sequences, see Fig. 3 and Fig. 4. Visually, we can see that a pattern appears in a sequence, if its itemsets are included in itemsets of the sequence in the same order and with a potential gap between them.

4 SEQuential patterns Scouting

Our first algorithm is called `SeqScout`. It is a sampling approach that exploits generalizations of database sequences, and searches for local optima w.r.t. the chosen quality measure. Fig. 5 provides an illustration for the method[3]. 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). Briefly, the idea of the UCB is to give a score to each sequence that quantifies an exploration-exploitation trade-off, and to choose the sequence with the best one (details on this part will be given later).

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

[3] 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. 5) 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.

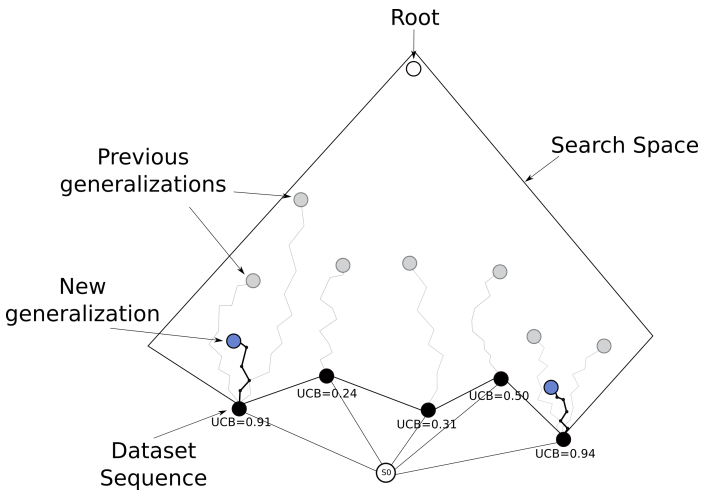

Fig. 5: Illustration of `SeqScout`.

scores of each sequence of the dataset, computed by using Equation 1 (see Section 4.1). $data_+$ contains the list of all sequences of the dataset 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 4.2) and its quality is computed (see Section 4.6). The created pattern is added to π (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 4.3). Finally, these patterns are processed thanks to a local optimization procedure (see Section 4.4). Moreover, `SeqScout` needs other modules that concern the selection of the quality measure (see Section 4.5) and the quality score computation (see Section 4.6).

4.1 SELECT Policy: 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 multi-armed bandit model is well known in the game theory literature [10]. 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 (exploitation), instead of trying others that could be more interesting in the long term (exploration)? Auer

et al. proposed a strategy called *UCB1* [3]. The idea is to give each arm a score, and to choose the one that maximizes it:

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

where $\bar{x_i}$ is the empirical mean of the ith arm, N_i is the number of plays of the ith 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 here the search space of extents, not the search space of all possible patterns. Exploring the search space of extents guarantees to find patterns with non null support while exploring the search space of all possible patterns leads towards many patterns with a null support. This is a crucial issue when dealing with sequences of itemsets.

4.2 ROLLOUT Policy: 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

Algorithm 1 SeqScout

```
function SeqScout(budget)
    π ← PriorityQueue()
    scores ← PriorityQueue()
    data+ ← FilterData()
    for all sequence in data+ do
        scores_ucb.add(sequence, ∞)
    end for
    while budget do
        seq, qual, N_i ← scores.bestUCB()
        seq_p, qual_p ← PlayArm(seq)
        π.add(seq_p, qual_p)
        scores.update(seq, (N_i*qual+qual_p)/(N_i+1), N_i + 1)
    end while
    π.add(OPTIMIZE(π))
    return π.topKNonRedundant()
end function

function OPTIMIZE(π)
    topK ← π.topKNonRedundant()
    for all pattern in topK do
        while pattern is not a local optima do
            pattern, qual ← BestNeighbor(pattern)
        end while
    end for
    return pattern, qual
end function
```

element. Indeed, most of the patterns in the search space have a null support [33]. `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:

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

The quality of the pattern is then computed, to update the UCB1 value of the sequence from which the pattern has been generated.

4.3 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]) that can be 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.

4.4 Local Optimum Search

Finally, a local optimum search is launched w.r.t. Definition 8. Various strategies can be used. The first possible strategy is the Steepest Ascend Hill Climbing [34]. 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 [34]: 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 [34]. 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 $|Neighborhood(s)|$, 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:

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

4.5 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, among others, the $F1$ score, informedness or the Weighted Relative Accuracy [23]. The later, the $WRAcc$, is commonly used for discriminant pattern mining and subgroup discovery. It compares the proportion of positive elements 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 solely based on pattern support in databases (whole dataset, or restricted to a class). It enables a number of optimizations. Using random draws makes it particularly difficult as each draw is independent: we cannot benefit from the same data structures as classical exhaustive pattern mining algorithms do (see, e.g., [4]).

4.6 Efficient Computation of Quality Scores

To improve the time efficiency of support computing, bitset representations have been proposed. For instance, SPAM uses a bitset representation of a pattern when computing an item- or set-extension at the end of a sequence [4]. In our case, we consider that an element can be inserted anywhere. Therefore, we use 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 of our strategy is given in Fig. 6. 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. The 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. 6). 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.

5 Monte Carlo Tree Search for Sequences

We now propose an extension of `SeqScout` that we call `MCTSExtent`. It is a significant evolution of `SeqScout` when looking towards a better trade-off between exploration and exploitation of a sampling strategy over the search space. It is based on the Monte Carlo Tree Search framework that is a founded evolution of

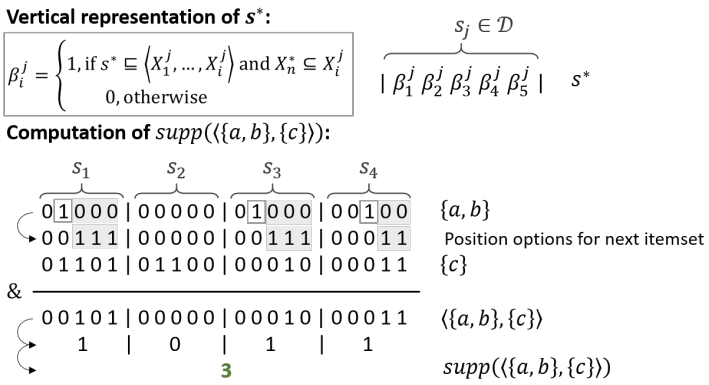

Fig. 6: Bitset representation and support computing.

bandit-based methods [9]. Like for `SeqScout`, our idea here is to explore the search space in a bottom-up way, contrary to classical search space exploration in pattern discovery. Indeed, instead of beginning the exploration by selecting general patterns, we start by isolating very specific patterns covering only few objects having the target label, and we construct better ones by adding other interesting objects of the database. In other terms, we directly explore groups of instances of the database, *i.e.*, extents with at least one positive element.

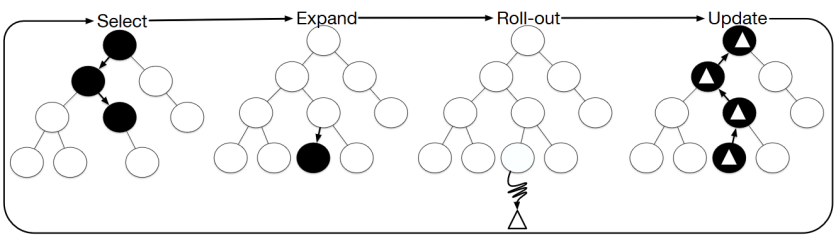

Fig. 7: Steps of Monte Carlo Tree Search, inspired by [9]

.

5.1 Background

Monte Carlo Tree Search is a method originally used in games to explore vast search spaces following an exploration-exploitation trade-off. The idea is to sequentially sample the search space to get information about locations of promising areas (meaning good game configurations in the case of games), to guide the search. At each iteration, the algorithm starts from the initial node, navigates through already discovered nodes, until it finds an interesting one (SELECT).

Then it computes a specialization of the selected node (EXPAND), samples the search space from this expanded node (ROLLOUT) and gives back the information about the quality of the sampled area (UPDATE). The search can be stopped anytime. A schema of the four steps of MCTS is given in Fig. 7.

5.2 `MCTSExtent` Description

The pseudo-code of `MCTSExtent` is given in Algorithm 2. The main loop of the algorithm runs as long as a computational budget is available (Lines 4-11) making the algorithm anytime. Note that each node of the MCTS tree contains a list of positive instances, in other words its extent, and a pattern covering them and only them.

The first step is to SELECT a node (Lines 15-23), *i.e*, choosing the best node in the tree following the exploration-exploitation trade-off w.r.t. UCB value. This step helps to guide the search towards promising areas of the search space, having good quality patterns, without ignoring that other non-explored parts can also be interesting. Thus, at each iteration, the algorithm checks if the current node is fully-expanded, according to Definition 10. If not, it is selected, and if yes, the SELECT procedure continues.

Definition 10 (Fully-expanded Node) A fully-expanded node is a node which has already been expanded in all possible ways. It means there are no positive sequences to add to its extent to compute an unseen Longest Common Subsequence (LCS).

Then, a new node is created with EXPAND (Lines 25-29) as follows:

- a new positive database object is added to the extent of the selected node;
- the Longest Common Subsequence (see Subsection 5.4) between this object and the pattern of the selected node is computed;
- the extent of this LCS is then computed.

We need to perform this last step because the LCS can cover more objects of the database than the union of the previous extent and the new positive object. It enables to get one of the most specialized pattern covering at least selected node objects and the new positive object. Notice that its computational cost is negligible compared to support computation. Moreover, it creates a pattern having positive elements: it can lead to the creation of a good quality pattern if it covers less negative elements.

The next step is the ROLLOUT (Lines 31-38), where the node is generalized the same way as explained in Subsection 4.2. Finally, the score of the ROLLOUT is used to update the quality of the path followed to reach it: the expanded node, the selected node, and all its parents until the root. This step can be seen as a back-propagation of the result: the search space has been sampled, and we update the quality of nodes of the tree to indicate if this area is interesting or not for the next iterations.

Finally, once the time budget is reached, the algorithm returns the top-k non-redundant elements, i.e., it performs the same redundancy filtering step as `SeqScout` (see Subsection 4.3).

Algorithm 2 MCTSExtent

```
function MctsExtent(budget)
    π ← PriorityQueue()
    create s_0 empty root having all instances as children
    while computational budget do
        s_sel ← Select(s_0)
        s_exp, qual_exp ← Expand(s_sel)
        s_roll, Δ ← Rollout(s_exp)
        Update(s_exp, Δ)
        π.add(s_exp, qual_exp)
        π.add(s_roll, Δ)
    end while
    return π.topKNonRedundant()
end function

function Select(s)
    while s is not root do
        if s is not fully-expanded then
            return s
        else
            s ← BestChild(s)
        end if
    end while
end function

function Expand(s)
    s_+ ← randomly choose a positive instance not in s
    s_exp ← extent(LCS(s, s_+))
    return s_exp, reward_{s_exp}
end function

function Rollout(s)
    for item in each itemset in s do
        if random > 0.5 then
            s.remove(item)
        end if
    end for
    return s, reward_s
end function

function Update(s, Δ)
    while s ≠ s_0 do
        Q(s) ← (N(s)*Q(s)+Δ) / (N(s)+1)
        N(s) ← N(s) + 1
    end while
end function

function BestChild(s)
    return argmax_{s' in children of s} UCB(s, s')
end function
```

5.3 Example

An example of `MCTSExtent` steps is given in Figure 9. First, the SELECT function starts from the root node and it selects the best node to expand thanks to UCB. Node containing Object 3 is not fully expanded, so it is selected. Then the EX-

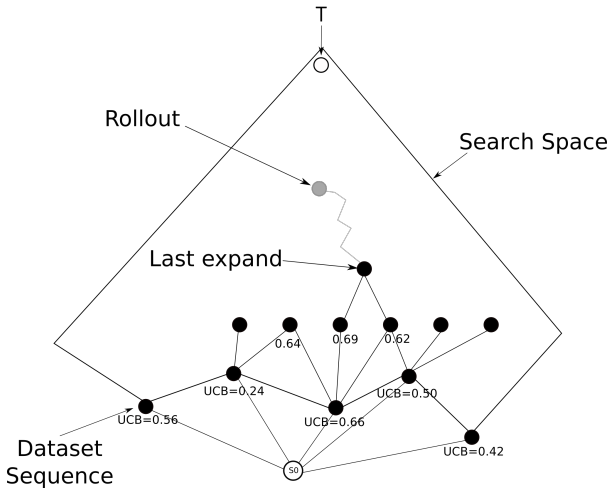

Fig. 8: `MCTSExtent` Principle.

PAND function first takes a random positive element, which is 2 in this case, and adds it to the node. Then the LCS between the pattern of the selected node and Object 2 is computed: the extent of the node is now {1,2,3}. From this node, we can now make a ROLLOUT and the pattern is generalized. Finally, nodes from the path are updated with the reward of the ROLLOUT to provide feedback about the quality of this area of the search space.

5.4 Computing a Longest Common Subsequence

`MCTSExtent` needs the classical concept of Longest Common Subsequence (LCS). Hirschberg et al. have described a dynamic programming algorithm that solves this problem in polynomial time for sequences of items [21]. However, it does not work on sequences of itemsets. Vlachos et al. [37] introduced an algorithm for sequences of multidimensional real-values items, having parameters to enforce constraints on the difference between real values. This is a different problem. Egho et al. have proposed an algorithm to compute the number of distinct common subsequences between two sequences of itemsets [17]. Such an algorithm does not generate the needed longest common subsequence.

Theorem 2 *Let two sequences of itemsets S^1 and S^2 of size n and m. We denote $S^1_{\leq i}$ the prefix of S_1, i.e, $S^1_{\leq i} = \langle X_1 ... X_i \rangle$. Let $LCS(S^1, S^2)$ be the set of the longest common subsequences of S_1 and S_2, or more formally:*

$$LCS(S^1, S^2) = \underset{s \in CS(S^1, S^2)}{\text{argmax}} \; length(s)$$

with length(s) the length of s, and $CS(S^1, S^2)$ the set of all common subsequences between S^1 and S^2.

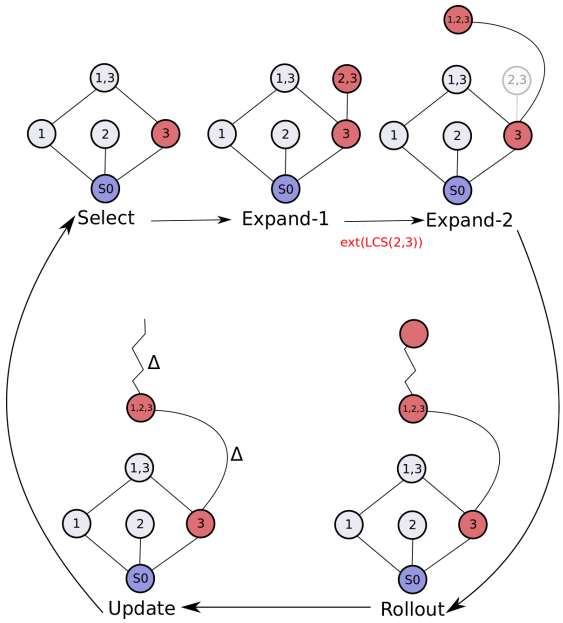

Fig. 9: Illustration of `MCTSExtent`.

We then have:

$$LCS(S^1_{\leq i}, S^2_{\leq j}) = \underset{s \in \chi}{\operatorname{argmax}}\, length(s)$$

with

$$\chi = \bigcup \left\{ \begin{array}{c} LCS(S^1_{\leq i-1}, S^2_{\leq j-1}) \cup (X^1_i \cap X^2_j) \ \ (CaseA) \\ \\ LCS(S^1_{\leq i-1}, S^2_{\leq j}) \ \ (CaseB) \\ \\ LCS(S^1_{\leq i}, S^2_{\leq j-1}) \ \ (CaseC) \end{array} \right\}$$

Note that it generalizes the theorem of LCS for sequences of items from [21], where $X^1_i \cap X^2_j$ is an itemset of size 1. If last items are equals, $LCS(S^1_{\leq i-1}, S^2_{\leq j})$ and $LCS(S^1_{\leq i}, S^2_{\leq j-1})$ are less or equally long than $LCS(S^1_{\leq i-1}, S^2_{\leq j-1})) + 1$, so it is not necessary to look at it to compute the LCS.

To prove this theorem, we will need the following lemma.

Lemma 1 $\forall s_1, s_2 \in S^2, s_1 \sqsubseteq S^1,\ s_2 \sqsubseteq S^2$*:*

$$LCS(s_1, s_2) \in CS(S^1, S^2)$$

This comes from the fact that $LCS(s_1, s_2) \sqsubseteq s_1$ and $LCS(s_1, s_2) \sqsubseteq s_2$, so by transitivity, $LCS(s_1, s_2) \sqsubseteq S^1$ and $LCS(s_1, s_2) \sqsubseteq S^2$.

Proof Reductio ad absurdum: Let us assume we have an LCS which is not in a case of this theorem. We are not in Case B, a $LCS(S^1_{\leq i-1}, S^2_{\leq j})$. The LCS can then finish with an item of X^1_i. Let us assume this is the case for this demonstration.

Symmetrically, it can finish with an item of X_j^2 for Case C.
The considered LCS then finishes with an itemset composed of elements from $X_i^1 \cup X_j^2$ (it must be common, by definition). As it must be the longest, the last itemset of this LCS is $X = X_i^1 \cap X_j^2$. We then have a LCS of the form $Y + X$, with $Y \notin LCS(S^1_{\leq i-1}, S^2_{\leq j-1})$, because we cannot be in the Case A. There is then only two possibilities for Y: whether Y is not common to S^1 and S^2, or Y is smaller than $LCS(S^1_{\leq i-1}, S^2_{\leq j-1})$. In both cases, it violates the definition of LCS. Thus, we showed by contradiction that:

$$LCS(S^1_{\leq i}, S^2_{\leq j}) \subseteq A \cup B \cup C. \tag{2}$$

Knowing that $S^1_{\leq i-1} \sqsubseteq S^1_{\leq i}$ and given Lemma 1, we can derive that

$$LCS(S^1_{\leq i-1}, S^2_{\leq j}) \in CS(S^1_{\leq i}, S^2_{\leq j}) \tag{3}$$

Symmetrically,

$$LCS(S^1_{\leq i}, S^2_{\leq j-1}) \in CS(S^1_{\leq i}, S^2_{\leq j}) \tag{4}$$

$$LCS(S^1_{\leq i-1}, S^2_{\leq j-1}) \in CS(S^1_{\leq i}, S^2_{\leq j}) \tag{5}$$

$$X_i^1 \cap X_j^2 \in CS(S^1_{\leq i}, S^2_{\leq j}) \tag{6}$$

We can deduce from (3), (4), (5) and (6) that:

$$A \cup B \cup C \subseteq CS(S^1_{\leq i}, S^2_{\leq j}) \tag{7}$$

From (2) and (7), we can conclude that the theorem is proven. □

The pseudo-code of the dynamic programming procedure computing a LCS of two sequences of itemsets is presented in Algorithm 3. First the matrix C is filled with 0. Then, we use a bottom-up approach to fill the matrix with correct values, using the previous theorem. Note that a i, j cell contains the length of $LCS(S^1_{\leq i-1}, S^2_{\leq j-1})$. Once the computation of the length of the LCS is done, a backtracking procedure is launched, to construct the solution (Lines 1-21). We begin by looking at the "bottom-right" of the matrix (Line 32). We then check if there is an intersection between itemsets i and j (Line 5). If this is the case, we check if the sub-problem at rank $i-1$ or $j-1$ have the same LCS (those cases are here to check what path the LCS procedure took). Else, we add the intersection to the LCS, and we jump to the sub-problem of size $i-1$, $j-1$. If the intersection is null, we go to the sub-problem $i-1$ or $j-1$ having the maximum LCS (Lines 17-21). The procedure stops if we reach a sub-problem of 0 (Lines 2-3).

An example is given in Figure 10. The matrix is filled from the top-left cell to the bottom-right. At each step, following the theorem, we take the maximum value between the left cell, the upper cell, and the cell in the upper-left diagonal plus the length of the intersection of current itemsets.

Complexity Let l_1 and l_2 be the length of S^1 and S^2. Let X_{max} be the largest itemset present in the dataset. The computing of each cell of the matrix requires to look at 3 cells and to compute the intersection of two itemsets. This operation has a complexity of $O(3+|X_{max}|)$, so the time complexity of LCS is $O(l_1 * l_2 * |X_{max}|)$. The worst case of the backtracking procedure is $O(l_1 + l_2)$, which is negligible comparing to the complexity of LCS. The space complexity is $O(l_1 * l_2)$.

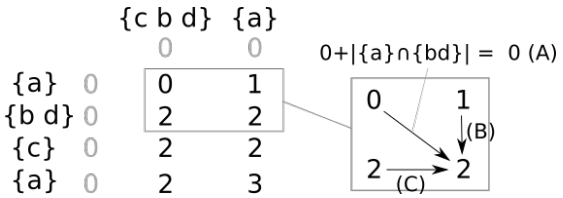

Fig. 10: An example of the dynamic programming for LCS

Algorithm 3 LCS

```
function BACKTRACK_LCS(C, S^1, S^2, i, j, lcs)
    if i = 0 or j = 0 then
        return
    end if
    inter ← S^1_i ∩ S^2_j
    if inter ≠ ∅ then
        if C(i − 1, j) = C(i, j) then
            return backtrack_LCS(C, S^1, S^2, i − 1, j, lcs)
        end if
        if C(i, j − 1) = C(i, j) then
            return backtrack_LCS(C, S^1, S^2, i, j − 1, lcs)
        else
            lcs.insert(0, inter)
            return backtrack_LCS(C, S^1, S^2, i − 1, j − 1, lcs)
        end if
    else
        if C(i, j − 1) > C(i − 1, j) then
            return backtrack_LCS(C, S^1, S^2, i, j − 1, lcs)
        else
            return backtrack_LCS(C, S^1, S^2, i − 1, j, lcs)
        end if
    end if
end function

function LCS(budget)
    Initialize C with dimensions size(S^1) ∗ size(S^2) filled with 0's
    for i=1 to size(S^1) + 1 do
        for j=1 to size(S^2) + 1 do
            inter ← S^1_i ∩ S^2_j
            C(i, j) ← max(C(i − 1, j − 1) + length(inter), C(i − 1, j), C(i, j − 1))
        end for
    end for
    final_lcs ← list()
    backtrack_LCS(C, S^1, S^2, length(S^1), length(S^2), final_lcs)
    return final_lcs
end function
```

6 Experiments

Let us now discuss an extensive empirical evaluation of `SeqScout` and `MCTSextent`. First, we present our evaluation protocol in terms of: (1) the datasets used (see Section 6.1), including our original dataset on the popular "Rocket League" video game (see Section 6.2), and (2) the baseline algorithms we use for comparison (see

Section 6.3). We then report the results of our evaluation using multiple datasets in Sections 6.5-6.14, and provide qualitative assessment of the performance of our proposed algorithms on Starcraft II and Rocket League datasets in Sections 6.15 and 6.16, respectively. All the experiments were performed on a machine equipped with Intel Core i7-8750H CPU and 16GB RAM. Algorithms mentioned hereafter are implemented in Python 3.6. The source code of all algorithms and experiments are available online[4].

6.1 Datasets

We used several popular benchmark datasets to evaluate the behavior of our algorithms, namely **promoters** [14], **context** [14], **splice** [14] and **skating** [29].

We also apply our algorithms on the real life dataset **sc2** that has been used in [8]. It was extracted from Starcraft II games. Starcraft II is a Real Time Strategy (RTS) game that is well known within the AI community. Recently, it has attracted more attention after the publication of the Google DeepMind AlphaStar results, an AI defeating human players [11]. 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.

We also use **jmlr**, a dataset consisting of abstracts of articles published in the Journal of Machine Learning Research [36]. In this dataset, we consider sequences of words. As class labels, we used the occurrence of the word "svm" in a sequence, *i.e.*, we label by "+" sequences of words containing the word "svm", removing words after it, and "-" for others.

Finally, we propose an **original** dataset **rl** for Rocket League[5] game analytics. Notice that e-sport will be considered for the first time at the 2021 Olympic Games in Tokyo and a Rocket League tournament with nation-based teams will be organized at this occasion.

Table 3 summarizes the statistics of the used datasets.

6.2 The Rocket League Use Case

Rocket League is a video game where each player controls a "rocket-powered" car on a football field. The goal is then to score more than the adverse team by hitting a ball into their goal. Each of the two teams can be composed of 1, 2 or 3 players. The game is very competitive, with international competitions and professional players. Each player is ranked using a score increasing with victories and decreasing with defeats.

[4] `https://github.com/Romathonat/MCTSExtent`

[5] `https://www.rocketleague.com/`

Table 3: Datasets

Dataset	$\|\mathcal{D}\|$	$\|\mathcal{I}\|$	l_{max}	Search Space Size
promoters [14]	106	4	57	$1.59 * 10^{41}$
context [14]	240	47	123	$5.25 * 10^{224}$
splice [14]	3,190	8	60	$3.29 * 10^{62}$
sc2 [8]	5,000	30	30	$6.48 * 10^{48}$
skating [29]	530	41	120	$1.16 * 10^{212}$
jmlr [36]	788	3,836	228	$1.84 * 10^{853}$
rl	148	16	157	$2.77 * 10^{212}$

One of the particularity of the game is the lack of semantics of the player actions during the game. Indeed, players control their car by using very basic instructions like "accelerate", "turn left", "jump", "slide", and these sequences of instructions can create special "skills". These skills are easily recognizable by spectators or analysts, but the game (we mean its computer program) cannot currently recognize them. Note that such a problem does not occur for a game like Starcraft II where, for instance, construction of a certain building implies a single user's action, which is, therefore, associated to an explicit semantics (however discovery of players strategies is similar). It is also important to note here that the same player will not generate exactly the same sequence of controls two times for the same skill. Indeed, positions of the ball and player's car vary and it is impossible for a human to be perfectly regular on a skill, her timing varies, and there are a lot of micro-adjustments to the trajectory of the car. This is illustrated in Fig. 1 and Fig. 2. Each sequence represents an instance of the same shot (the "ceiling shot"), performed by the same player. Clearly, sequences from the same class can be very different, making the discovery of interesting patterns difficult. In other words, there is a lot of *noise* in collected data. However, sequences of controls pressed by the player obviously hide patterns related to particular skills (see pattern p in Fig. 3 and Fig. 4 for example), and that is exactly what we want to discover here.

Extracting patterns from sequences of input is interesting to better understand what players are performing, e.g., in terms of skills/figures, for example in the process of learning from examples. Those patterns could also be used as features for a classification system to automatically reward players who perform skills, or to improve the score of the player as some skills are difficult to perform.

To generate the dataset, we implemented a key logger to get inputs of the controller of the player. We then asked her to perform some skills among 6 different ones. After doing a skill, we labelled the generated sequence by the selected skill. The goal is then to extract pattern discriminative of a target skill. Note that here we are in the case of sequences of itemsets as the player can press different controls at the same time. For example, she can use "boost" and "jump" to make her car fly in the air. In our study, we use a controller with sixteen buttons.

The schema given in Fig. 11 illustrates how a sequence is constructed for a controller with two buttons `a` and `b`. The black lines correspond to the time the buttons are pressed. We use three filling colors to depict three possible combinations of buttons pressed, namely: (1) blue, when button `a` is pressed, (2) yellow, when button `b` is pressed, and (3) red, when `a` and `b` are pressed together. The corresponding itemsets are given underneath. The dataset is available online with the rest of our work.

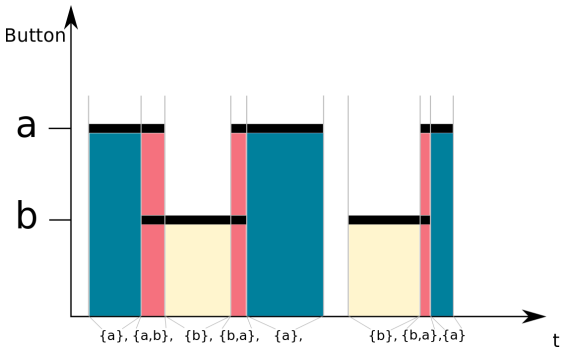

Fig. 11: A sequence generated by two buttons

6.3 Baselines

To the best of our knowledge, we are the first to address the problem of discriminative pattern mining in sequences of itemsets, and therefore, there are not available algorithms that can be used directly as baselines for the evaluation. However, there are several algorithms for processing sequences of items. Therefore, to evaluate `SeqScout` and `MCTSExtent`, we have modified two algorithms, namely `misère` [16] and `BeamSearch` [24], such that they process sequences of itemsets.

First, we implemented an extension of `misère` [16], the original version of which was handling sequences of events only but not sequences of itemsets. Second, we implemented `BeamSearch` as a sequence-oriented version of a beam search algorithm. 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]. Moreover, to ensure fair comparisons, we removed the post-processing optimization of `SeqScout` that is studied more precisely in Section 6.14.

6.4 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 as such objective measures need to compute 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 larger one with the same time budget is not relevant.

6.5 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 computed with `misère`, `BeamSearch`, `SeqScout` and `MCTSExtent`. Fig. 12 provides absolute results. `MCTSExtent` is clearly the best solution on each dataset. Interestingly, we can note that `BeamSearch` is sometimes inefficient (see on **splice**).

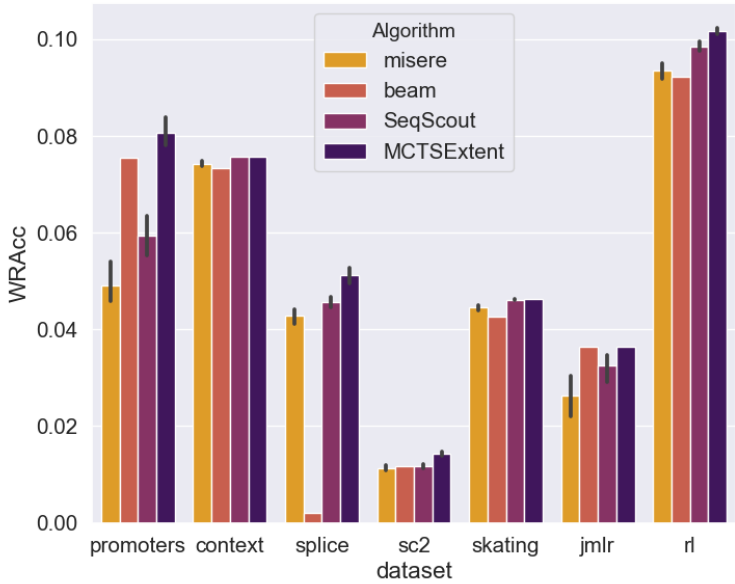

Fig. 12: Mean $WRAcc$ of top-5 best patterns (10K iterations)

We can clearly see that `MCTSExtent` and `SeqScout` perform particularly well on **splice** compared to `BeamSearch`. Overall, `MCTSExtent` provides better results.

To achieve a comprehensive evaluation, we fixed a relatively small time budget of 60 seconds to compare the performances of algorithms, i.e., the limiting factor here is not the number of iterations but the given time budget. Results can be seen in Fig. 13. `MCTSExtent` generally outperforms other algorithms in terms of average $WRAcc$. We can also note that similarly to `misère`, `SeqScout` shows a significant decrease of performance on **jmlr**. Indeed, it seems that the strategy of taking a sequence and generalizing it is not efficient in a short time budget on this dataset. In contrast, `MCTSExtent` guides the search toward promising patterns more quickly, resulting in better performances.

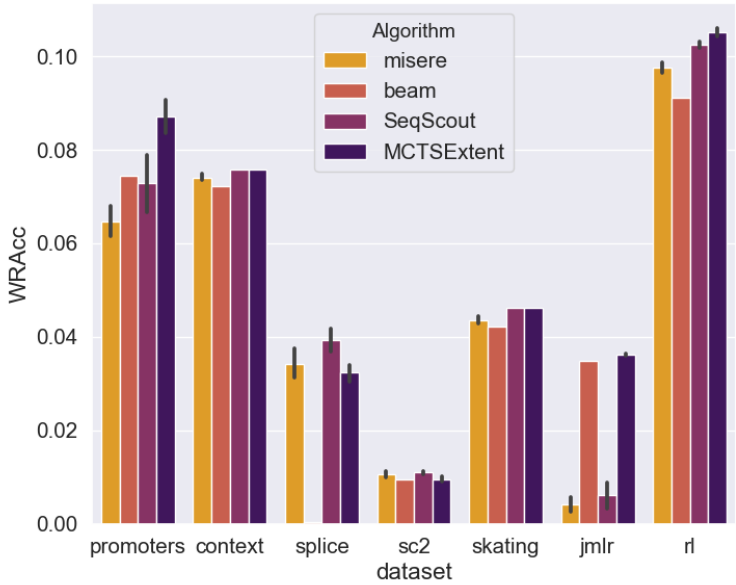

Fig. 13: Mean $WRAcc$ of top-5 best patterns (time budget: 60s)

6.6 Quality w.r.t. Number of Iterations

We show the result quality in terms of $WRAcc$ over the number of iterations. Fig. 14-20 depict the results for the top-5 non-redundant patterns on each dataset. Note that for the same data, the results may vary from run to run, due to the random nature of `misère` and `SeqScout`. It explains some fluctuations of the quality. Nevertheless, for each $iteration_num$ setting, `MCTSExtent` has shown better results.

6.7 Using other Quality Measures

To empirically illustrate the measure agnostic characteristic of `SeqScout` and `MCTSExtent`, we have used other quality measures, namely $F1$-score and $Informedness$. The results are shown in Table 4. Our algorithms generally give better results.

6.8 Performance Study under Varying θ

We also evaluate performances of the algorithms when varying the value of the similarity threshold θ. Fig. 21 shows the performance on the dataset **context** (results are similar on other datasets). We did not include the results for $\theta = 0$ because it would mean finding patterns with totally disjoint extents. It results

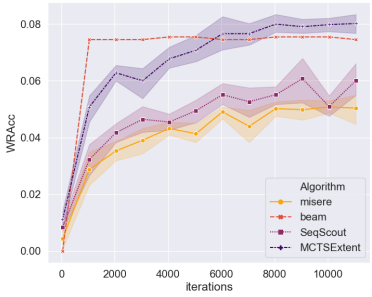

Fig. 14: Average $WRAcc$ for top-5 patterns w.r.t. iterations (**promoters**)

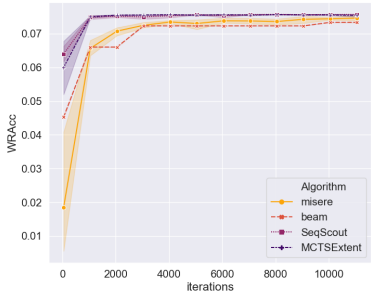

Fig. 15: Average $WRAcc$ for top-5 patterns w.r.t. iterations (**context**)

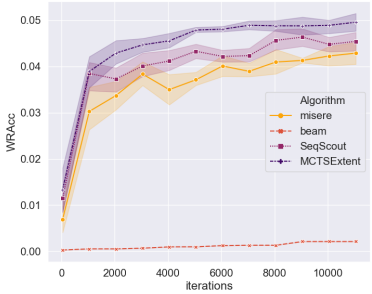

Fig. 16: Average $WRAcc$ for top-5 patterns w.r.t. iterations (**splice**)

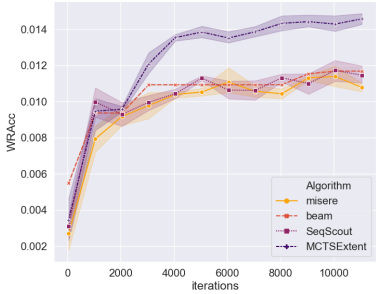

Fig. 17: Average $WRAcc$ for top-5 patterns w.r.t. iterations (**sc2**)

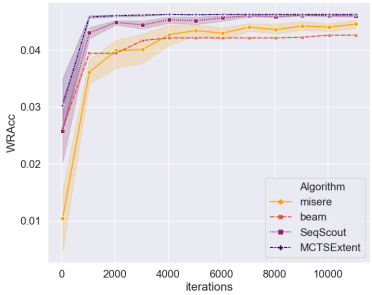

Fig. 18: Average $WRAcc$ for top-5 patterns w.r.t. iterations (**skating**)

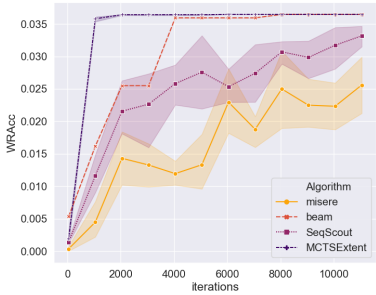

Fig. 19: Average $WRAcc$ for top-5 patterns w.r.t. iterations (**jmlr**)

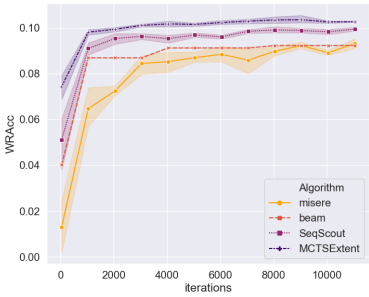

Fig. 20: Average $WRAcc$ for top-5 patterns w.r.t. iterations (**rl**)

Table 4: Mean values of measures for top-5 patterns.

Dataset	Informedness				F1			
Algorithm	misere	BeamS.	SeqScout	MCTSExtent	misere	BeamS.	SeqScout	MCTSExtent
promoters	0.081	0.089	0.088	**0.144**	0.600	0.545	0.565	**0.636**
context	0.465	0.462	0.470	**0.472**	0.581	0.569	**0.586**	**0.586**
splice	0.373	0.041	0.392	**0.397**	0.428	0.086	**0.452**	0.451
sc2	0.013	0.006	0.011	**0.015**	0.531	0.533	0.541	**0.550**
skating	0.419	0.389	**0.423**	**0.423**	0.391	**0.402**	0.393	**0.402**
jmlr	0.439	**0.545**	0.449	**0.545**	0.330	0.402	0.337	**0.421**
rl	0.697	0.689	0.758	**0.779**	0.697	0.726	**0.739**	0.733

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 relative performances of algorithms are approximately the same for all θ values.

6.9 Performance Study under Varying k (top-k)

We investigate the performance of the search for top-k patterns when changing the k parameter. Fig. 22 shows the results when considering the **sc2** dataset (the behaviour is similar on other datasets). `MCTSExtent` gives better results. Note that the mean $WRAcc$ decreases for all algorithms, as increasing k leads to the selection of lower quality patterns.

6.10 Quality vs. Search Space Size

We evaluate the average $WRAcc$ of top-5 patterns w.r.t. the maximum length of sequences (see Fig. 24). To do so, we have truncated the dataset to control the maximum lengths of sequences. We demonstrate it on the **rl** dataset. The plot shows that `MCTSExtent` gives generally better mean $WRAcc$ values whatever the search space size is. We also note a general increase of quality when increasing the search space size. Indeed, some patterns that are bad for a smaller maximum length can appear in positive elements for larger maximum lengths, resulting in an increasing quality of patterns. Note that the opposite phenomenon can also appear.

6.11 Sequence Lengths

The pattern lengths on all datasets are reported in Fig. 23. Let us consider the **splice** dataset: `BeamSearch` gives short patterns (max 8), which is significantly less than `MCTSExtent`, `SeqScout` and `misère`. This may explain why the `BeamSearch` result quality is bad on this dataset (see Fig. 12). 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. Another interesting observation is that on **jmlr**, the length of sequences is 1 or 2. This means that interesting patterns of this dataset are short ones, and are in fact itemsets: that is why `BeamSearch`

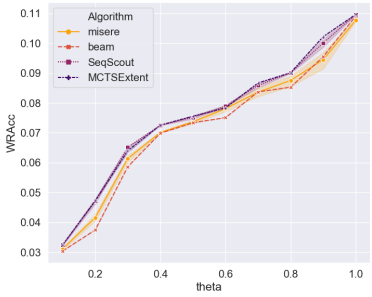

Fig. 21: Mean $WRAcc$ of top-5 patterns vs. similarity threshold θ (**context**)

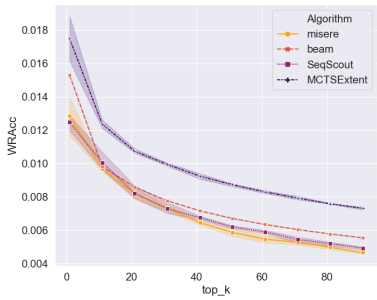

Fig. 22: Mean $WRAcc$ of top-k patterns vs. k (**sc2**)

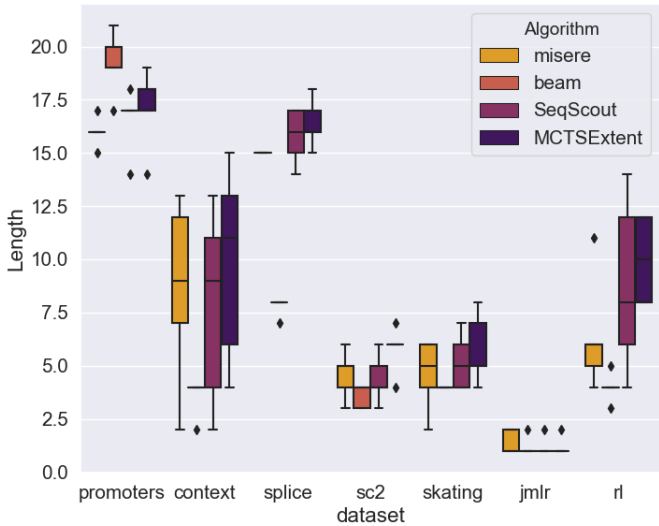

Fig. 23: Length of top-5 best patterns - 10K iterations

performs very well on it. In fact, using a subgroup discovery algorithm exploiting itemset descriptions should give similar results.

6.12 Non Diversified `BeamSearch`

Leeuwen et Al. [24] proposed to filter redundant patterns during the `beam-search` procedure. We wanted to check if this strategy remains efficient in the case of sequences of itemsets. In fact, the main issue with the classical `BeamSearch` approach is that when filtering is done in post-processing, we can get less than k patterns, depending on the configuration. In fact, a classical `BeamSearch` is not

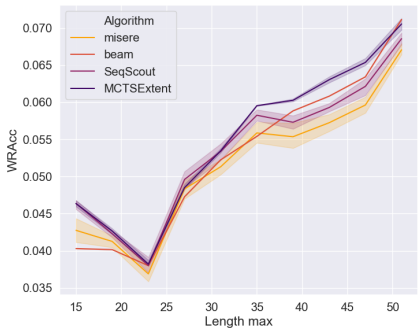

Fig. 24: Mean $WRAcc$ w.r.t. the maximum length of sequences on **rl**.

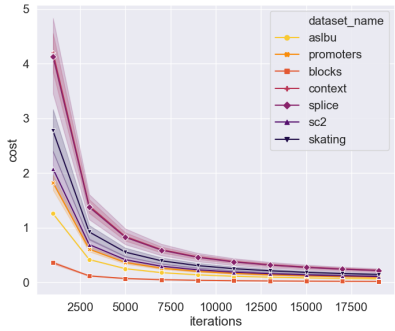

Fig. 25: Additional cost of local optima search

BeamSearch	Diversified	Non-Diversified
promoters	0.075	X
context	0.073	0.073
splice	0.002	0.002
sc2	0.116	0.119
skating	0.043	0.044
jmlr	0.036	0.036
rl	0.092	0.099

Table 5: $WRAcc$ for top-5 patterns on diversified vs. non-diversified `BeamSearch`

relevant to solve our problem. A comparison of the two `BeamSearch` strategies is given in Table 5. We added a "X" when the number of found patterns is lower than k.

6.13 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 6. We can see that the bitset representation tends to give a performance gain for datasets with smaller search space size upper bound, but leads to a decrease of performances for the majority of the datasets. For example, **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.

Table 6: Number of iterations in Bitset vs. Integer set representation

Dataset	Integer Set	Bitset	Variation(%)
promoters	7,185	8,858	24
context	4,651	2,667	-47
splice	289	254	-12
sc2	943	605	-36
skating	4,001	1,283	-68
jmlr	704	31	-95
rl	8839	7799	-12

6.14 Local Optima Search

The local optima search uses more iterations. This over cost depends on the dataset. In Fig. 25, 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 (also referred to as the cost). The more iterations we have, the more negligible the ratio is. 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 large, leading to a very long local optima search (110,000 iterations for 5 patterns in our experiments). Consequently, we note that the local optima search may not be the relevant choice with this kind of dataset.

We also added it as a post-processing step to each of our algorithm to compare general quality increase depicted in Fig. 26-29. As we can see, the more initial iterations are given, the better the mean quality is, and the lesser the local optima search is interesting. This emphasizes on the fact that `MCTSExtent` performs a good exploration of the search space. Note that on some dataset, like **promoters**, this local search generally leads to an important quality increase.

6.15 Qualitative Evaluation on **sc2**

As `MCTSExtent` clearly gives the best results, let us now discuss the quality of a discovered pattern extracted from the dataset **sc2**. The question that we aim at answering can be formulated as follows. Given a faction, what are the best strategies to win? In other words, `MCTSExtent` will look for patterns of construction which are characteristic to the victory of this faction.

For example, let us consider the "Terran" faction. One of the best computed patterns is: $\langle\{Hatchery(Z)\},\{Supply(T)\},\{Factory(T)\},\{Supply(T)\}\rangle$. In this example, we use the colors as indicators of fractions, *i.e.*, blue for "Terran" and purple for "Zerg". We can see that the "Terran" player is investing in military units ($\{Supply(T)\}$ and $\{Factory(T)\}$). The "Zerg" player chooses to invest in its economy by creating a $\{Hatchery(T)\}$: 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 [8].

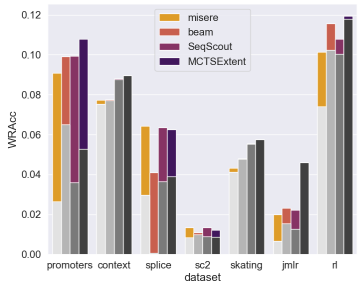

Fig. 26: Added value of local optima search for 1,000 iterations

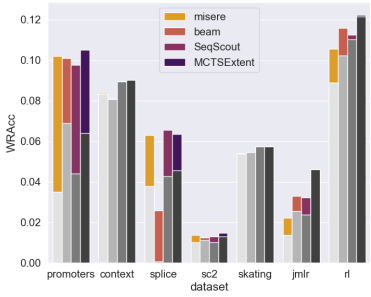

Fig. 27: Added value of local optima search for 3,000 iterations

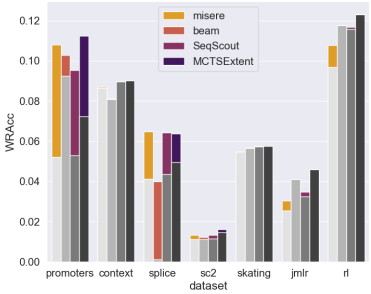

Fig. 28: Added value of local optima search for 6,000 iterations

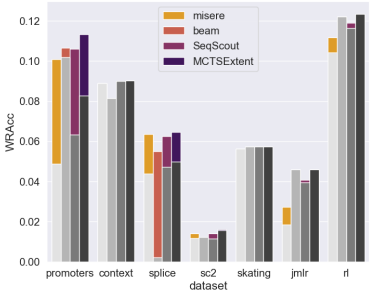

Fig. 29: Added value of local optima search for 10,000 iterations

6.16 Qualitative Evaluation on **rl**

Looking at extracted patterns for the original **rl** dataset can also validate the relevance of our algorithm. We recall that each sequence of the dataset is composed of inputs of the player, and the class to the name of the skill she performed. Sequence lengths are between 20 and 150. Moreover, as there is a lot of noise in the data due to human behaviour (micro-adjustments of trajectory, difficulty to replay exactly the same sequence with same timing), finding discriminative patterns of a skill is challenging.

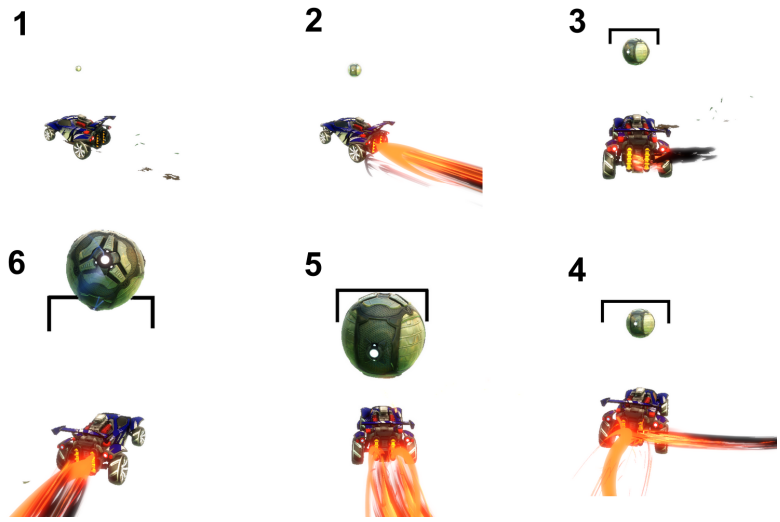

Fig. 30: Decomposition of the "slide shoot"

Interestingly, `MCTSExtent` can cope with this problem. In the experiment, we selected one target skill at a time. First, we focused on what we call the "**slide shoot**". The sequence of movements of a car is described in Fig. 30. The top-3 patterns obtained are:

1. $\langle\{boost\},\{slide\},\{right\ boost\},\{boost\}\rangle$ ($WRAcc = 0.102$),
2. $\langle\{boost\},\{slide\},\{left\ boost\},\{boost\}\rangle$ ($WRAcc = 0.101$),
3. $\langle\{boost\},\{left\},\{slide\},\{boost\}\rangle$ ($WRAcc = 0.098$).

This example shows the interest of looking at a pattern *set*. Indeed, this skill can be executed in two ways: by coming from the left part of the ball and sliding to the right, or coming from the right and sliding to the left. These top-3 patterns reflect this: the first one corresponds indeed to a right slide while the second one corresponds to a left slide. Note that we removed the item "accelerate" which is non informative because Rocket League players use it nearly all the time: it is present in 79.3% of itemsets.

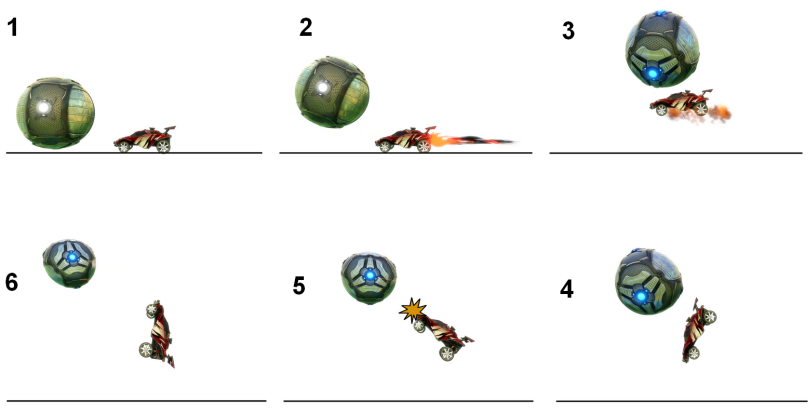

Fig. 31: Decomposition of the "musty flick"

Another popular skill when playing Rocket League is called the **musty flick**. The sequence of movements of the car is described in Fig. 31. The top-3 patterns obtained by `MCTSExtent` are:

1. $\langle\{boost\},\{boost\},\{up\ jump\},\{jump\ down\}\rangle$ ($WRAcc = 0.131$),
2. $\langle\{boost\},\{up\},\{up\ jump\},\{jump\ down\}\rangle$ ($WRAcc = 0.131$),
3. $\langle\{up\ jump\},\{down\}\rangle$ ($WRAcc = 0.125$).

As we can see, those sequences, particularly the first one, correspond indeed to the performed skill. Note that commands are inverted on the vertical axis: pressing "up" makes the car orientate to the ground.

We can also get other useful informations from those patterns. We can see that the second pattern is interesting because it represents a part of the skill, but is not sufficient to reproduce it. However, it is discriminative of it in this dataset, because this sequence of inputs is present only in this skill, leading to the well-known conclusion that *data quality and variety is a top priority to extract useful knowledge*.

Moreover, one may notice that the first and the second patterns are quite similar. However, the non-redundancy constraint garantees us that the extents of each of the found patterns are not too much overlapping. We can deduce that the player tends to initiate the down orientation of the car at the same time that she jumps more often than she does it before jumping. Those examples show that *having a set of patterns is more relevant than using a single one*. For the same skill, there are variations that can be captured.

7 Conclusion

We have discussed the problem of finding dicriminative patterns in labeled sequences of itemsets. We have presented two algorithms `SeqScout` and `MCTSExtent`

to discover relevant subgroups in sequences of itemsets. Though we are not aware of available algorithms to solve the same problem, we have implemented the adaptations of two other algorithms, namely `misère` and `BeamSearch`, such that they could be applied for the case of sequences of itemsets. This has been useful to support our empirical evaluation.

Furthermore, to implement `MCTSExtent`, we have also introduced a new algorithm to compute a Longest Common Subsequence between two sequences. Our experiments have shown that `MCTSExtent` outperforms all other algorithms, without the need for additional parameter tuning as needed in the case of `Beam Search`.

Moreover, we have created a new dataset, containing sequences of inputs of a player of Rocket League video game, labeled with particular skills she performed. We have used this dataset in our evaluation procedure, together with other various benchmark datasets.

Our future work will aim at exploring the application of `MCTSExtent` to other pattern languages and contexts. Indeed, the method of `MCTSExtent` can be easily adapted to other pattern languages, by redefining the "rollout" step (or generalization) and the "Common pattern" step, i.e., the Longest Common Subsequence here. Finally, in the context of game analytics, we will focus on the classification of players' actions, which are often recognizable only by human game experts, and on grasping a better understanding of the system in general by extracting and interpreting valuable patterns [27].

References

1. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proceedings IEEE ICDE 1995, pp. 3–14 (1995). DOI 10.1109/ICDE.1995.380415
2. Atzmüller, M., Puppe, F.: SD-map: A fast algorithm for exhaustive subgroup discovery. In: Proceedings PKDD 2006, pp. 6–17 (2006). DOI 10.1007/11871637_6
3. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2), 235–256 (2002). DOI 10.1023/A:1013689704352
4. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: Proceedings ACM SIGKDD 2002, pp. 429–435 (2002). DOI 10.1145/775047.775109
5. Belfodil, A., Belfodil, A., Kaytoue, M.: Anytime subgroup discovery in numerical domains with guarantees. In: Proceedings ECML/PKDD 2018 (2), pp. 500–516 (2018). DOI 10.1007/978-3-030-10928-8_30
6. Boley, M., Lucchese, C., Paurat, D., Gärtner, T.: Direct local pattern sampling by efficient two-step random procedures. In: Proceedings ACM SIGKDD 2011, pp. 582–590 (2011). DOI 10.1145/2020408.2020500
7. Bosc, G., Boulicaut, J.F., Raïssi, C., Kaytoue, M.: Anytime discovery of a diverse set of patterns with monte carlo tree search. Data Min. Knowl. Discov. **32**(3), 604–650 (2018). DOI 10.1007/s10618-017-0547-5
8. Bosc, G., Tan, P., Boulicaut, J.F., Raïssi, C., Kaytoue, M.: A Pattern Mining Approach to Study Strategy Balance in RTS Games. IEEE Trans. Comput. Intellig. and AI in Games **9**(2), 123–132 (2017). DOI 10.1109/TCIAIG.2015.2511819
9. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Trans. Comput. Intellig. and AI in Games **4**(1), 1–43 (2012). DOI 10.1109/TCIAIG.2012.2186810
10. Bubeck, S., Cesa-Bianchi, N.: Regret analysis of stochastic and nonstochastic multi-armed bandit problems. Foundations and Trends in Machine Learning **5**(1), 1–122 (2012). DOI 10.1561/2200000024. URL `http://dx.doi.org/10.1561/2200000024`
11. DeepMind: Alphastar: Mastering the real-time strategy game StarCraft II (2019). URL `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`

12. Diop, L., Diop, C.T., Giacometti, A., Li, D., Soulet, A.: Sequential pattern sampling with norm constraints. In: Proceedings IEEE ICDM 2018, pp. 89–98 (2018). DOI 10.1109/ICDM.2018.00024
13. Diop, L., Diop, C.T., Giacometti, A., Li, D., Soulet, A.: Sequential pattern sampling with norm-based utility. Knowl. Inf. Syst. (2019). DOI 10.1007/s10115-019-01417-3. URL `https://doi.org/10.1007/s10115-019-01417-3`
14. Dua, D., Karra Taniskidou, E.: UCI machine learning repository (2017). URL `http://archive.ics.uci.edu/ml`
15. Duivesteijn, W., Feelders, A.J., Knobbe, A.: Exceptional model mining. Data Min. Knowl. Discov. **30**(1), 47–98 (2016). DOI 10.1007/s10618-015-0403-4
16. Egho, E., Gay, D., Boullé, M., Voisine, N., Clérot, F.: A user parameter-free approach for mining robust sequential classification rules. Knowl. Inf. Syst. **52**(1), 53–81 (2017). DOI 10.1007/s10115-016-1002-4
17. Egho, E., Raïssi, C., Calders, T., Jay, N., Napoli, A.: On measuring similarity for sequences of itemsets. Data Min. Knowl. Discov. **29**(3), 732–764 (2015). DOI 10.1007/s10618-014-0362-1
18. Fradkin, D., Mörchen, F.: Mining sequential patterns for classification. Knowl. Inf. Syst. **45**(3), 731–749 (2015). DOI 10.1007/s10115-014-0817-0
19. Giacometti, A., Li, D.H., Marcel, P., Soulet, A.: 20 years of pattern mining: a bibliometric survey. SIGKDD Explor. Newsl. **15**(1), 41–50 (2013). DOI 10.1145/2594473.2594480
20. Gsponer, S., Smyth, B., Ifrim, G.: Efficient sequence regression by learning linear models in all-subsequence space. In: Proceedings ECML/PKDD 2017 (2), pp. 37–52 (2017). DOI 10.1007/978-3-319-71246-8_3
21. Hirschberg, D.S.: Algorithms for the longest common subsequence problem. Journal of the ACM **24**(4), 664–675 (1977). DOI 10.1145/322033.322044
22. Ji, X., Bailey, J., Dong, G.: Mining minimal distinguishing subsequence patterns with gap constraints. Knowl. Inf. Syst. **11**(3), 259–286 (2007). DOI 10.1007/s10115-006-0038-2
23. Lavrac, N., Flach, P.A., Zupan, B.: Rule evaluation measures: A unifying view. In: Proceedings ILP 1999, pp. 174–185 (1999). DOI 10.1007/3-540-48751-4_17
24. Leeuwen, M.V., Knobbe, A.J.: Diverse subgroup set discovery. Data Min. Knowl. Discov. **25**(2), 208–242 (2012). DOI 10.1007/s10618-012-0273-y
25. Letham, B., Rudin, C., Madigan, D.: Sequential event prediction. Machine Learning **93**(2), 357–380 (2013). DOI 10.1007/s10994-013-5356-5
26. Mathonat, R., Boulicaut, J.F., Kaytoue, M.: SeqScout: Using a bandit model to discover interesting subgroups in labeled sequences. In: Proceedings IEEE DSAA 2019, pp. 81–90 (2019)
27. Mathonat, R., Boulicaut, J.F., Kaytoue, M.: A behavioral pattern mining approach to model player skills in rocket league. In: 2020 IEEE Conference on Games (CoG) (2020)
28. Moens, S., Boley, M.: Instant exceptional model mining using weighted controlled pattern sampling. In: Proceedings IDA 2014, pp. 203–214 (2014). DOI 10.1007/978-3-319-12571-8_18
29. Mörchen, F., Ultsch, A.: Efficient mining of understandable patterns from multivariate interval time series. Data Min. Knowl. Discov. **15**(2), 181–215 (2007). DOI 10.1007/s10618-007-0070-1
30. Novak, P.K., Lavrač, N., Webb, G.I.: Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. Journal Machine Learning Research **10**, 377–403 (2009). DOI 10.1145/1577069.1577083
31. Nowozin, S., Bakir, G., Tsuda, K.: Discriminative subsequence mining for action classification. In: Proceedings IEEE ICSV 2007, pp. 1–8 (2007). DOI 10.1109/ICCV.2007.4409049
32. Pei, J., Han, J., Mortazavi-asl, B., Zhu, H.: Mining access patterns efficiently from web logs. In: T. Terano, H. Liu, A.L.P. Chen (eds.) Knowledge Discovery and Data Mining. Current Issues and New Applications, pp. 396–407. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
33. Raïssi, C., Pei, J.: Towards bounding sequential patterns. In: Proceedings ACM SIGKDD 2011, pp. 1379–1387 (2011). DOI 10.1145/2020408.2020612
34. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall Press, Upper Saddle River, NJ, USA (2009)
35. She, R., Chen, F., Wang, K., Ester, M., Gardy, J.L., Brinkman, F.S.L.: Frequent-subsequence-based prediction of outer membrane proteins. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '03, p. 436–445. Association for Computing Machinery, New York, NY, USA (2003). DOI 10.1145/956750.956800. URL `https://doi.org/10.1145/956750.956800`

36. Tatti, N., Vreeken, J.: The long and the short of it: Summarising event sequences with serial episodes. Proceedings ACM SIGKDD 2012 pp. 462–470 (2012). DOI 10.1145/2339530.2339606

37. Vlachos, M., Hadjieleftheriou, M., Gunopulos, D., Keogh, E.: Indexing multi-dimensional time-series with support for multiple distance measures. In: Proceedings ACM SIGKDD 2003, pp. 216–225. ACM (2003). DOI 10.1145/956750.956777

38. Wrobel, S.: An algorithm for multi-relational discovery of subgroups. In: Proceedings PKDD 1997, pp. 78–87 (1997)

39. Xing, Z., Pei, J., Keogh, E.: A brief survey on sequence classification. SIGKDD Explor. Newsl. **12**(1), 40–48 (2010). DOI 10.1145/1882471.1882478

40. Zaki, M., Lesh, N., Ogihara, M.: Planmine: Predicting plan failures using sequence mining. Artif. Intell. Rev. **14**, 421–446 (2000). DOI 10.1023/A:1006612804250

41. Zaki, M.J.: Spade: An efficient algorithm for mining frequent sequences. Machine Learning **42**(1), 31–60 (2001). DOI 10.1023/A:1007652502315

42. Zhou, C., Cule, B., Goethals, B.: Pattern based sequence classification. IEEE Trans. Knowl. Data Eng. **28**, 1285–1298 (2016). DOI 10.1109/TKDE.2015.2510010

43. Zimmermann, A., Nijssen, S.: Supervised pattern mining and applications to classification. Frequent Pattern Mining pp. 437–439 (2014)

Romain Mathonat received his Ph.D. degree in computer sciences at INSA Lyon in the DM2L team (Data Mining and Machine Learning) of LIRIS Lab in 2020. His research interests include supervised rule discovery, knowledge discovery and data science in general. He took interest in different application domains like urban farming, predictive maintenance or game analytics.

Diana Nurbakova Diana Nurbakova received the BS and MSc degrees in Applied Mathematics and Informatics, and MA degree in Linguistics and Traductology from Perm State University (Perm, Russia) in 2009, 2011 and 2011, respectively. She also received the MSc degree in Computational Science from the University of Reading (Reading, UK) in 2011. She received the PhD degree in Computer Science from the National Institute of Applied Sciences of Lyon (Lyon, France) in 2018. Apart from the experience in academia, she has 4.5-year experience of work in the industry in Russia and France, including R&D Project Manager position at Tilkee company (France). She is currently an Associate Professor of Computer Science at the National Institute of Applied Sciences of Lyon (Lyon, France) and a member of LIRIS research laboratory. Her research interests lie in the general area of artificial intelligence and data analysis, especially in user modelling, user behaviour pattern mining, recommender systems.

Jean-François Boulicaut is currently professor of computer science at INSA Lyon, and was the director of its Computer Science and Information Technology department between 2015 and 2018. After a sabbatical year at the departement of computer science in the university of Helsinki (Finland), he came back to France and has been in 1998 the founding leader of a Data Mining research group at INSA Lyon. He is now a senior member of theDM2L group (Data Mining and Machine Learning) within the large Research Unit LIRIS UMR CNRS 5205. His own expertise concerns the design of constraint-based mining algorithms (e.g., supporting pattern discovery like rules or sequential patterns, subgroup discovery, clustering and co-clustering). He has been a member of the Data Mining and Knowledge Discovery journal editorial board from 2006 until 2016 and has served in the program committees of every major data mining conference.

Mehdi Kaytoue has been an associate professor of Computer Science at INSA de Lyon (France) since 2012. He received his Ph.D. in Computer Science from the Université de Lorraine in 2011 (Nancy, France). His main research interests are artificial intelligence, data-mining, and especially pattern mining with formal concept analysis. Original methods and algorithms developed as part of his research are applied to various domains with a particular attention to olfaction in neuroscience, social media analysis, electronic sports and video game analytics. In 2018, he joined the company Infologic R&D, software editor of Copilote, an enterprise resource planning (ERP), to lead the data and AI strategy.