Mining Balanced Patterns in Real-Time Strategy Games*

LIRIS Research Report

Guillaume Bosc[1], Mehdi Kaytoue[1,4], Chedy Raïssi[2]
Jean-François Boulicaut[1] and Philip Tan[3]

[1]Université de Lyon, CNRS, INSA-Lyon, LIRIS, UMR5205, F-69621, France
[2]LORIA (CNRS - Inria NGE - Université de Lorraine), Vandoeuvre-lès-Nancy, France
[3]Creative Director, MIT Game Lab, Cambridge (MA), USA
[4]Universidade Federal de Minas Gerais, Belo Horizonte, Brasil
Contact: guillaume.bosc@insa-lyon.fr
May 2014

Abstract

Given a set of class-labeled objects taking descriptions from a partially ordered set (e.g. itemsets, graphs, intervals, etc.), the task of finding description generalizations that strongly cover a class and weakly the others, has attracted a lot of attention in artificial intelligence (machine learning and data-mining) under various names (hypothesis, contrast sets, subgroups, emerging/jumping patterns, etc.). We propose in this paper to discover sequential patterns that discriminate victory from zero-sum games. We present efficient algorithms and show how emerging pattern mining may be used to balance games (an important issue in the game industry) by using it to analyze strategies with a novel measure, the balance measure. We experiment with the designed algorithms on a real-world strategy game played professionally as an electronic sport.

1 Introduction

Discovering patterns that highly distinguish a dataset from others is an important task in artificial intelligence [14]. One of the main reason is that such patterns enable the building of comprehensible classifiers [9]. In the general settings, we are given a set of objects of different classes that take descriptions, generally from a partially ordered set (itemsets, graphs, intervals, etc.) [12]. The goal is to find good descriptions generalizations, that mostly appear in one class of objects and not in the others. In different fields of AI and applied mathematics, such descriptions have different names [14] like *version spaces* [13], *contrast sets* [2] and *subgroups discovery* [8] in machine learning, *emerging patterns* [6] in data-mining; or *no counter-example hypothesis* in formal

*This article is an extended version an article [3] presented at the European Conference on Artificial Intelligence (ECAI 2014). It contains supplementary materials such as algorithm descriptions and extended experiments. This work has been partially supported by the FAPEMIG/INRIA project "Mining Data guided by Knowledge Models".

concept analysis [12]. In the present work, we choose a pattern mining oriented discussion [1]. Emerging patterns are the descriptions which are frequent in a class of objects and much less frequent in the rest.

Many real-world data are dynamically generated and can be represented as sequential data where each object is described by a sequence of sets of events. Over the last decade, many algorithms and methods have been proposed to extract different types of sequential patterns and effective pattern representations. Sequential patterns are used in web search, customer trajectory analysis, workflow analysis, and bioinformatics. For frequent sequential patterns as well, their articulation as local knowledge can be used to build classifiers in various real-world applications with emerging patterns [7, 14, 16].

In this work, we are interested in discovering sequences from zero-sum games databases. The particularity is that they represent traces of real-time (and not turn-by-turn) games opposing two players with only one possible winner. Our goal is to provide the basic pattern mining tools to exhibit characteristics such as weaknesses and strengths of players. More precisely, the question is how to define a measure that gives for an arbitrary sequence of actions a notion of balance. Intuitively, consider the example where a player chooses the sequence of actions s and the opponent s' (s, s' forms a single game): what are the odds of playing s against s'? In other terms, a game s, s' is well-balanced when playing s against s' allows to win as much as it allows to loose[1]. This is a challenging task as current state-of-the-art methods fail to handle this specific type of sequences. We show indeed that a balance measure can be easily defined as a growth rate measure (that characterizes emerging patterns) for a particular case of zero-sum games: those where both players do not have the same set of available actions. In the general case, this approach fails and new settings that we develop are required.

From an application point of view, sequential patterns from zero-sum games allow analysts to derive statistics on the success of a player with different tactics for real-time strategy games. These video games are becoming the *de facto* sensations in organized competitions, especially between professionals competitors taking part in international tournaments, with prize-pools in millions of US$, surrounded by managers, teams and sponsors [17] and followed by a large world-wide fan base [11]. As such, balancing the games is one of the most important issues of any E-Sport to ensure a fair competitive aspect on which depends a complete economic model. Surprisingly, few pattern mining approaches have been published [10] about this new data play ground that interests both the game industry, social sciences researchers and computer scientists.

The rest of the paper is organized as follows. Section 2 recalls the basics of sequential pattern mining. Section 3 introduces the problem of mining non-zero games sequences as balanced patterns, while sections 4 and 5 gives two methods for achieving such a task. Algorithms are then designed (Section 6) and experimented with E-Sport data (Section 7) before to conclude.

2 Preliminaries

Let $\mathcal{I}$ be a finite set of *items*. Any non-empty subset $X \subseteq \mathcal{I}$ is called an *itemset*. A *sequence* $s = \langle X_1, ..., X_l \rangle$ is an ordered list of $l > 0$ itemsets. l is the length of the

[1]Note that such a balance measure can be seen as a utility function in zero-sum games where s, s' is a strategy profile. Sequential patterns can be helpful for building normal or extensive forms representation of a game to be analyzed with game theory tools. This is indeed a difficult problem for real time games where the search space can be extremely large. This however goes beyond the scope of this paper.

Table 1: A sequence database $\mathcal{D}$.

id	$s \in \mathcal{D}$	$class(s)$
s_1	$\langle a\{abc\}\{ac\}d\{cf\}\rangle$	$+$
s_2	$\langle\{ad\}c\{bc\}\{ae\}\rangle$	$+$
s_3	$\langle\{ef\}\{ab\}\{df\}cb\rangle$	$-$
s_4	$\langle eg\{af\}cbc\rangle$	$-$

sequence, whereas $\sum_{i=1}^{l} |X_i|$ is its size. Considering $\mathcal{I}$ as a set of events (or actions), an itemset denotes simultaneous events while the order between two itemsets a strict preceding relation. A sequence database $\mathcal{D}$ is a set of $|\mathcal{D}|$ sequences over $\mathcal{I}$. Sequences may have different lengths and sizes and are uniquely identified, see Table 1 (omitting the third column).

Definition (subsequence). A sequence $s = \langle X_1, ..., X_{l_s}\rangle$ is a *subsequence* of a sequence $s' = \langle X'_1, ..., X'_{l'_s}\rangle$, denoted $s \sqsubseteq s'$, if there exists $1 \leq j_1 < j_2 < ... < j_{l_s} \leq l'_s$ such that $s_1 \subseteq s'_{j_1}, s_2 \subseteq s'_j, ..., s_{l_s} \subseteq s'_{j_{l_s}}$.

Definition (Support and frequency) Let $\mathcal{D}$ be a sequence database. The *support* of a sequence s is given by $support(s, \mathcal{D}) = \{s' \mid s \sqsubseteq s',\ s' \in \mathcal{D}\}$, and the frequency by $freq(s, \mathcal{D}) = |support(s, \mathcal{D})|/|\mathcal{D}|$.

Problem (Frequent sequential pattern mining). Given a minimal frequency threshold $0 < \sigma \leq 1$, the problem is to find all sequences s such as $freq(s, \mathcal{D}) \geq \sigma$: the frequent sequential patterns.

In several situations, each sequence of $\mathcal{D}$ is associated to a class label. Let $class : \mathcal{D} \to \{+, -\}$ a mapping that associates to each sequence a positive or negative label (hence two classes). $\mathcal{D}$ is accordingly partitioned in two bases, with the positive (resp. negative) sequences $\mathcal{D}^+$ (resp. $\mathcal{D}^-$) and $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$, $\mathcal{D}^+ \cap \mathcal{D}^- = \emptyset$. The growth-rate characterizes the discriminating power of a pattern for one class w.r.t. the other [4, 6, 14][2].

Definition (Growth-rate). Given a sequence database $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$, the growth-rate of a sequential pattern from $\mathcal{D}^x$ to $\mathcal{D}^y$ ($x \neq y$ and $x, y \in \{+, -\}$), is given by

$$growth\_rate(s, \mathcal{D}^x) = \frac{|support(s, \mathcal{D}^x)|}{|\mathcal{D}^x|} \times \frac{|\mathcal{D}^y|}{|support(s, \mathcal{D}^y)|}$$

Example. Let $\mathcal{D} = \{s_1, s_2, s_3, s_4\}$ with $\mathcal{I} = \{a,\ b,\ c,\ d,\ e,\ f,\ g\}$ be a sequence database given in Table 1. To ease reading, we drop the commas, but also braces for singletons. For a given sequence $s = \langle abc\rangle$, we have $s \sqsubseteq s_1$, $s \sqsubseteq s_4$, $s \not\sqsubseteq s_2$, $s \not\sqsubseteq s_3$. With $\sigma = \frac{3}{4}$, $\langle acc\rangle$ is frequent, $\langle a\{bc\}a\rangle$ is not. Consider now the class label from $\{+, -\}$ associated to each database sequence, we have $growth\_rate(\langle cb\rangle, \mathcal{D}^-) = \frac{2}{2} \times \frac{2}{1} = 2$, i.e. $\langle cb\rangle$ is two times more present in class $-$ than in class $+$.

3 Introducing the Problem of Strategy Elicitation

We consider a competitive interaction as a sequence of actions made by two players where exactly one player wins (no ties), typically a zero-sum game. Such a sequential

[2]Note that non binary classes can be handled with slight modifications [4, 6].

game can be represented as a sequence of sets of actions, each action realized by one of the two players. When both players play in real-time, one can describe these interactions as sequences of itemsets. An itemset is then a set of simultaneous actions, or within an insignificant interval of time.

Definition (Interaction (sequence) database). Given a set of players $Players$ and a set of actions $Actions$, a sequence database $\mathcal{D}_{interaction}$ is called an *interaction database*. Each sequence denotes an interaction between two players and is defined over the set of items $\mathcal{I} = Actions \times Players$. A mapping $class : \mathcal{D}_{interaction} \rightarrow Players$ gives the winner of each interaction.

Example. In Table 2, the sequence identified s_1 can be interpreted as: "*Player p_1 did action a, then he did b and c while player p_2 did c, and finally p_1 did b while p_2 did d. At then end, the player p_1 wins*".

The problem here is to find sequences of actions of both interacting players (supposing that those actions are mutually dependent) as generalizations that appear frequently and to be able to characterize their discriminating ability for winning/loosing through a so-called *balance* measure. In sequential pattern mining settings, the goal is to find frequent sequences (strategies) and their balance (a growth-rate like measure). However, the notion of class has to be revisited to be able to handle winner and looser class labels, instead of the winning player. Indeed, intuitively, mining emerging patterns from an interaction database with the winning players as classes (as given in Table 2) does not fulfill our objectives. Therefore, we need to abstract the notion of player as either a winner or a looser. Now, consider a frequent sequence made only of actions $\langle a, b\rangle$, where a is made by the winner, and b by the looser. Consider also the same sequence, but this time a has been done by the looser and b by the winner. A ratio involving the support of both sequences can then give the balance of the sequence of actions $\langle a, b\rangle$. As such, we need to be able to find sequences of actions that are frequent, and for each action to know if it has been done by a winner or a looser.

A first idea is to split each interaction sequence into two sequences, one for each player, and label each positively (resp. negatively) when the given player has won (resp. has lost). This would lead to a sequence database as given in previous section (Table 1). However, there we loose the interaction dimension: an emerging pattern denotes a mostly winning (resp. loosing) sequence of action made by a winning (resp. loosing) player. Since actions of both players are mutually dependent in general, we wish that extracted patterns take into account actions of both players.

To solve this problem, we propose to differentiate two cases of interactions databases: (i) *non-mirror interaction databases* where both players have different (non-intersecting) sets of available actions; (ii) *mirror interaction databases* where both players can perform the same actions. We show that in the first case, emerging patterns as introduced in the literature (previous section) are able to answer the problem by slightly modifying the interaction database representation. In the second case, the most general (not dealt with in the literature to the best of our knowledge), we need new settings, and we propose to embed the class (positive or negative) in the definition of the items of a sequence, see Table 4. This is formalized in the two next sections, and it allows the design of efficient pattern mining algorithms in Section 6.

Table 2: An interaction sequence database $\mathcal{D}_{interaction}$

id	Interaction sequence	Winner
s_1	$\langle(p_1,a)\{(p_1,b)(p_1,c)(p_2,c)\}\{(p_2,a)(p_1,d)\}\rangle$	p_1
s_2	$\langle(p_3,a)\{(p_3,b)(p_3,c)(p_3,d)\}\{(p_1,b)(p_1,c)\}\rangle$	p_3

4 Balanced Patterns in Non-mirror Databases

In this section, we consider an interaction database, called *non-mirror database*, where the set of actions is different for each of the players in a single interaction. It means that we have only two types of players in each sequence and in the whole database, and those types are determined by the actions they can do. As such, the type can be also used as a winning class label. Typically, this happens for the strategy game we study in the experiment section. To characterize balanced patterns in such databases, we consider a simple transformation of the original interaction database $\mathcal{D}_{interactions}$ by dropping the player associated to each action, and labeling each sequence by the type of the winner. This allows us to express a balance measure as a growth-rate measure in this new data representation. The transformed database is then formally defined as follows.

Definition (Transformed interaction database). A sequence database $\mathcal{D}_t$ defined over the set of items (actions) $\mathcal{I}_1 \cup \mathcal{I}_2$ such as $\mathcal{I}_1 \cap \mathcal{I}_2 = \emptyset$ and $class : \mathcal{D}_t \rightarrow \{\mathcal{I}_1, \mathcal{I}_2\}$ is called a *transformed interaction database*.

Consider an interaction $s \in \mathcal{D}_t$ where the winner is characterized by the actions $\mathcal{I}_1$: we have $class(s) = \mathcal{I}_1$ that gives the winner of the interaction. This brings back the problem of finding frequent balanced interaction patterns to well-known emerging patterns settings. Indeed, consider an arbitrary pattern s over $\mathcal{I}_1 \cup \mathcal{I}_2$: it support in the whole database $support(s, \mathcal{D}_t)$ tells us its frequency, while $support(s, \mathcal{D}_t^{\mathcal{I}_1})$ and $support(s, \mathcal{D}_t^{\mathcal{I}_2})$ allows to define a balance measure as a growth-rate.

Mining balanced patterns from transformed interactions. Let $\mathcal{D}_t$ be a transformed database obtained from a non-mirror interaction database $\mathcal{D}_{interaction}$. $\mathcal{D}_t$ is defined over $\mathcal{I}_1 \cup \mathcal{I}_2$ where $\mathcal{I}_k$ represents the type k of player ($k \in \{1, 2\}$) and $class : \mathcal{D}_t \rightarrow \{\mathcal{I}_1, \mathcal{I}_2\}$ assigns to any sequence its winner type, and σ a minimum frequency threshold. The problem is to extract the set of so-called *frequent balanced patterns* $\mathcal{F}_t$ such as for any $s \in \mathcal{F}_t$, $freq(s, \mathcal{D}_t^{\mathcal{I}_1}) \geq \sigma$ and $freq(s, \mathcal{D}_t^{\mathcal{I}_2}) \geq \sigma$ (implying $freq(s, \mathcal{D}_t) \geq \sigma$) and the balance measure is computed and given by:

$$balance(s, \mathcal{D}_t^k) = \frac{|support(s, \mathcal{D}_t^k)|}{|support(s, \mathcal{D}_t^1)| + |support(s, \mathcal{D}_t^2)|}$$

The balance measure is a normalized version of the growth rate given in previous section so that $balance(.) \in]0, 1]$ and that $balance(s, \mathcal{D}_t^1) + balance(s, \mathcal{D}_t^2) = 1$ which entails a zero-sum game property. Note that we do not consider closed patterns [18]. Indeed, two patterns which have the same support may have different balance measure since a growth rate intuitively gives the proportion of appearance of the support in two different classes, and we do not seek patterns that maximize the growth-rate.

Example. Table 3 gives a transformed interaction database $\mathcal{D}_t$, obtained from a non-mirror interaction database $\mathcal{D}_{interaction}$, with $\mathcal{I}_1 = \{a, b\}$ and $\mathcal{I}_2 = \{c\}$ being the

Table 3: A non mirror interaction database

id	Interaction sequence	Winner
s_1	$\langle\{ab\}\{c\}\rangle$	$\mathcal{I}_1$
s_2	$\langle\{ab\}\{a\}\{c\}\rangle$	$\mathcal{I}_1$
s_3	$\langle\{abc\}\{c\}\rangle$	$\mathcal{I}_2$
s_4	$\langle\{ac\}\{a\}\{c\}\rangle$	$\mathcal{I}_1$
s_5	$\langle\{c\}\{b\}\{ac\}\rangle$	$\mathcal{I}_2$

Table 4: A signed interaction database.

id	$s \in \mathcal{D}_s$
s_1	$\langle a^+\{b^+c^-\}\rangle$
s_2	$\langle a^+\{b^+c^-\}c^+\rangle$
s_3	$\langle d^+\{b^+c^+\}d^-\rangle$
s_4	$\langle a^-\{b^+c^+\}\rangle$

sets of actions of each player's type. With $\sigma = 0.2$, $s = \langle\{ab\}\{c\}\rangle$ is a frequent balanced pattern since $freq(\mathcal{D}_t^{\mathcal{I}_1}, s) = 0.4$ and $freq(\mathcal{D}_t^{\mathcal{I}_2}, s) = 0.2$. Moreover, $balance(s, \mathcal{D}_t^{\mathcal{I}_1}) = \frac{2}{3}$ and $balance(s, \mathcal{D}_t^{\mathcal{I}_2}) = \frac{1}{3}$. This means that s wins two times more for the type 1 of player than for the type 2.

5 Balanced Patterns in Mirror Databases

In this section, we consider interaction sequence databases where the players have access to the same set of actions. Consequently, the latter cannot be partitioned in two sets and the previous approach fails. We propose a new interaction database representation, *signed interaction databases*, which allows to define *frequent balanced patterns* from an arbitrary interaction database (either mirror or non-mirror).

Definition (Signed interaction database). Recall that $Actions$ is a finite set of *actions* shared by both players. We introduce $\mathcal{I}_s = Actions \times \{+, -\}$ denoting actions associated either to a positive or negative class. A signed database $\mathcal{D}_s$ is built from an interaction database $\mathcal{D}_{interaction}$ as follows: Each action of an interaction sequence is signed $+$ if it is performed by the winner and signed $-$ if performed by the looser (both players and class labels are dropped).

Definition (Dual of an item, itemset and sequence). Let $\mathcal{I}_s = Actions \times \{+, -\}$ be the set of signed items, or actions. For any $(a, c) \in \mathcal{I}_s$, also written a^c, we define its *dual* as

$$dual(a, c) = dual(a^c) = (a, \{+, -\} \backslash c) = a^{\{+,-\}\backslash c}$$

Informally, it means that the dual of a signed action is the same action where the class c has changed. This definition is simply propagated for itemsets and sequences of

itemsets, for any $X \subseteq \mathcal{I}_s$ and any $s = \langle X_1, X_2, ... \rangle$ a sequence over $\mathcal{I}_s$:

$$\begin{aligned} dual(X) &= \{dual(x), \forall x \in X\} \\ dual(s) &= \langle dual(X_1), dual(X_2), ... \rangle \end{aligned}$$

Example. In Table 4, we have $\mathcal{I}_s = \{a, b, c, d\} \times \{+, -\}$, $dual(a^+) = a^-$ and $dual(s_1) = \langle a^-\{b^-c^+\}\rangle$.

These definitions allow us now to naturally introduce a balance measure that would, for a sequential pattern s give the proportion of its support among the support of both itself and its dual.

Definition (Balance measure). Let s be a frequent sequential pattern in a database $\mathcal{D}_s$. The balance measure of s is given by

$$balance(s) = \frac{|support(s, \mathcal{D}_s)|}{|support(s, \mathcal{D}_s)| + |support(dual(s), \mathcal{D}_s)|} \tag{1}$$

This intuitive definition however does not hold. Since actions are shared by the two players, both a sub-sequence and its dual may occur in a single sequence $s \in \mathcal{D}_s$. Consider the following example: $\mathcal{D}_s = \{\langle\{a^+b^-\}\{a^-b^+\}\rangle, \langle\{a^-b^+\}\rangle\}$ with $\sigma = \frac{1}{2}$. The sequence $s = \langle\{a^+b^-\}\rangle$ is a frequent sequential pattern, and $|support(s, \mathcal{D}_s)| = 1$. We have also $|support(dual(s), \mathcal{D}_s)| = |support(dual(\langle\{a^-b^+\}\rangle), \mathcal{D}_s)| = 2$. Hence, $balance(s) = \frac{1}{1+2} = \frac{1}{3}$. However, since s and $dual(s)$ both appear in the first sequence, it should not be counted two times. This leads us to the definition of the balance measure in the general case.

Definition (Generalized balance measure). For a sequential pattern s, the generalized balance measure is given by

$$balance_{gen}(s) = \frac{|support(s, \mathcal{D}_s) \backslash support(dual(s), \mathcal{D}_s)|}{|supportD(s, \mathcal{D}_s) \sqcup support(dual(s), \mathcal{D}_s)|} \tag{2}$$

where $\sqcup$ denotes the exclusive union $A \sqcup B = (A \cup B) \backslash (A \cap B)$. In the following, *balance* will always refer to the general version. We have that $balance(s) \in [0, 1]$ and $balance(s) + balance(dual(s)) = 1$ which expresses a zero-sum game property.

Mining balanced patterns from signed interactions. Let $\mathcal{D}_s$ be a signed interaction database defined over $\mathcal{I}_s$ generated from an interaction database $\mathcal{D}_{interaction}$, and σ a minimum frequency threshold. The problem is to extract the set of so-called *frequent balanced patterns* $\mathcal{F}_s$ such as for $s \in \mathcal{F}_s$, $freq_{\mathcal{D}_s}(s) \geq \sigma$, $freq_{\mathcal{D}_s}(dual(s)) \geq \sigma$ and the balance measure is computed and given by (2). Furthermore, the fact both s and $dual(s)$ have to be frequent leads to redundant information: it is enough to keep s along with its support, balance measure and intersection of support $common(s) = |support_{\mathcal{D}}(s) \cap support_{\mathcal{D}}(dual(s))|$ to know the measures of its dual. As such, the problem is also to compute a non redundant collection of patterns $\mathcal{F}_s$ where, if $s \in \mathcal{F}_s$ then $dual(s) \notin \mathcal{F}_s$.

Example. Table 4 gives a signed interaction database $\mathcal{D}_s$ obtained from an arbitrary $\mathcal{D}_{interaction}$. With $\sigma = \frac{1}{4}$, $s = \langle a^+c^-\rangle$ appears two times, its dual appear one time, hence $balance(s) = \frac{2}{3}$.

Remark 1. Any interaction database, mirror or non-mirror, can be represented as a signed interaction database. For non-mirror, one can easily prove that for any balanced

pattern s, we have $common(s) = \emptyset$ and thus Formula (1) is enough for computing its balance.

Remark 2. We consider a sequential pattern from $\mathcal{D}_s$ as a balanced pattern if and only if both s and $dual(s)$ are frequent (strictly frequent). One could leverage this definition by constraining only one of the two. The same applies in the formulation of the previous section. For sake of clarity, we consider only strictly frequent balanced patterns, since it would only slightly change the present definitions.

Finally, consider a non-mirror interaction database $\mathcal{D}_{interaction}$, as well as its transformed database $\mathcal{D}_t$ and its signed database $\mathcal{D}_s$. It can be naturally shown that both collections $\mathcal{F}_t$ and $\mathcal{F}_s$ are equivalent. This is because of the *non-mirror* property of the interaction database.

6 Algorithms

We discuss algorithms to extract frequent balanced patterns from interaction databases. We introduce first a well-known framework for mining frequent sequential patterns, called PATTERN-GROWTH and its associated algorithm PREFIXSPAN [15]. We adapt this framework in various ways to mine balanced patterns.

The PREFIXSPAN algorithm. Given a sequence database $\mathcal{D}$ over items $\mathcal{I}$ and a minimum frequency threshold σ, PREFIXSPAN outputs all frequent sequential patterns, and only them without generating any candidates [15]. Firstly, the database $\mathcal{D}$ is scanned one time to find all the frequent items from $\mathcal{I}$, called *size-1 sequential patterns*. Secondly, each of these *prefixes* is used to divide the search space: for one prefix, say $\langle a \rangle$ (and $a \in \mathcal{I}$), one retains only sequences of $\mathcal{D}$ containing a and only keeps for each of these sequences the longest suffix starting by a. The set of the remaining sequences is called a *projected database* w.r.t. to prefix $\langle a \rangle$, written $\mathcal{D}_{|\langle a \rangle}$. Thirdly, this projected database is scanned to generated the *size-2 sequential patterns* having $\langle a \rangle$ as prefix. The process is recursively applied leading to a tree structure where each node represents a frequent sequential pattern (associated with a projected database of a least $\sigma \times |\mathcal{D}|$ sequences) and an edge to an extension: the item added to a size-k sequential pattern to generate a size-$(k+1)$ sequential pattern. For a prefix s and an item a, two kinds of extensions are considered: appending a as a new suffix itemset of s, noted $s \circ_s a$, and appending a within the last itemset of s, written $s \circ_i a$ ($\circ$ denotes an extension in general). At the end, the pattern tree structure is explored and each node outputs a pattern.

Mining Balanced Patterns with EMERGSPAN. Given a sequential database over an arbitrary $\mathcal{I}$, where each sequence is labeled by a class (among two classes) and a minimum frequency threshold, the general problem here is to find all frequent patterns, each one provided with its growth-rate. In our settings, this involves mining a non-mirror interaction database $\mathcal{D}_{interaction}$ as a transformed database $\mathcal{D}_t$ where sequences are composed of actions of both players and the class label gives the winner's type. The balance measure is then exactly the growth rate as defined in the general settings. It has been shown in [16] that computing the growth-rate can be done as a post-processing of a sequential pattern mining algorithm and that this post-processing is negligible w.r.t. to the whole extraction time. With PREFIXSPAN, it works as follows. For each sequence of the database $\mathcal{D}_t$, its class is appended as a new itemset at its end. It ensures that frequent patterns containing a class label are leaves of the pattern tree, and that

two sequential patterns that differ only by their classes have the same direct parent. Computing frequent balanced patterns is then straightforward with a single depth-first traversal of the pattern tree (only direct parents of leaves are considered when having two children obtained both by an extension of a class label).

Mining Balanced Patterns with PREFIXSPANNAIVE**.** Now we consider a signed interaction database $\mathcal{D}_s$ over $\mathcal{I}_s$ and a minimal frequency threshold σ. The problem is to extract the set of frequent sequential patterns provided with their balanced measure. We can first use the original PREFIXSPAN algorithm to build the frequent pattern tree structure. We need for each pattern to (i) compute its balance, and (ii) to ensure that for a pattern and its dual only one of them is output. We then propose to store for each node (equivalently sequential pattern) a pointer to its dual pattern. At the first level of the tree, a node represents a single frequent item, and there is a pointer towards its dual (if both are frequent). Recursively, when a item is used to expand a sequential pattern p to obtain a pattern q, we compute the pointer towards $dual(q)$ by searching among the children of $dual(p)$. If $dual(q)$ exists, the pattern q is output, $dual(q)$ is flagged as already output (redundant pattern) and the process recursively continues. Otherwise a backtracking is operated. In this way, it never happens that both q and $dual(q)$ are output, we extract the set of all frequent balanced patterns (ensuring that for a pattern q both q and $dual(q)$ are frequent). Computing the balance for non mirror databases is straightforward since for each node/pattern we have access to its dual support. For mirror databases, we need however to know $common(q)$ which is stored for each node. The proof of the completeness and correctness of PREFIXSPANNAIVE for extracting all balanced patterns without redundancy is trivial: first, PREFIXSPAN extracts all frequent patterns, thus any pattern s and its dual $dual(s)$ are nodes of the pattern tree and none can be missed; second, as we visit in the tree traversal both a pattern s and its dual $dual(s)$ (if frequent), we ensure no redundant patterns.

Mining Balanced Patterns with BALANCESPAN**.** The problem of PREFIXSPAN-NAIVE is that it generates both a pattern and its dual as different nodes in the pattern tree. Furthermore, it also generates nodes for patterns that are frequent but whose dual is not frequent. Consequently, and this is shown in the experiments (Section 7), an important amount of nodes are useless. To solve that problem, and to be sure only nodes corresponding to balanced patterns are generated (and only them, i.e. correct and complete), we propose the BALANCESPAN approach. The general idea is the following: instead of considering each item $a \in \mathcal{I}_s$ as an extension on sequence s leading to a new projected database $\mathcal{D}_{s|s \circ a}$ and consequently a new node in the pattern tree, we consider simultaneously an item and its dual, hence two projected databases $\mathcal{D}_{s|s \circ a}$ and $\mathcal{D}_{s|s \circ dual(a)}$ (this is done for both kinds of extensions $\circ_i$ and $\circ_s$). Thus, it ensures us that no redundant patterns are generated, since both a sequence s and $dual(s)$ are generated at the same node, and it allows to compute $balance(s)$ (or $balance(dual(s))$ directly if and only if both s and $dual(s)$ are frequent. It follows that BALANCESPAN produces a correct and complete collection of frequent balanced patterns. Note that dealing with Remark 2 is then straightforward if one is interested to produce non-strict frequent patterns: the recursion continues until both projected databases do not produce frequent patterns.

7 Experiments

We experiment our approach to assess its computational feasibility and, helped by a high level StarCraft II player, the quality of the extracted patterns.

7.1 Context and data description

E-Sport and competitive gaming. Professional level competitors in video games have emerged over the last few years. These *"cyber-athletes"* take part in international tournaments with increasing prize-pools, surrounded by managers, teams and sponsors [17], and a world-wide fan base follows them in great numbers on social TVs [11]. "Electronic sport" should attract industrial actors and researchers in the next few years around video game analytics, just like with traditional sports. We study one of the most competitive real-time strategy games (RTS) [11], StarCraft II (Blizzard Entertainment, 2010) which has its own world-wide players ranking system (ELO) and annual world cup competition series (WCS) with a US$1.6 millions prize pool for the year 2013. StarCraft II is a real time strategy game where a game involves two players each choosing a faction among Zerg (Z), Protoss (P) and Terran (T): there are 6 different possible match-ups with different strategies of game. During a game, two players are battling on a map (aerial view), controlling buildings and units to gather supply, build an army with the final goal of winning by destroying the opponent's forces[3]. Such actions (training, building, moving, attacking) are done in real-time. Each faction (Z,P,T) allows different units and buildings with distinctive weaknesses and strengths following a rock-paper-scissors principle. A strategy is hidden in large sequences of actions generated by players and called *replays*.

Datasets. StarCraft II replays are files that store any action made by all players during a game, and are easily accessible on the Web[4]. We retained $91,503$ games with a total of 3.19 years of game time. The average length of a game is about 20 minutes. Games were selected if implying very high level players (in the highest leagues and playing at least 200 actions per minutes), since casual (by opposition to professional) players are not able to follow specific strategies. We divided the $91,503$ replays into six different sequence datasets, one for every match ups (since there are 3 factions). Buildings are one of the key elements of a strategy, since they allow different kinds of units production: from each replay, we derive a sequence where the items represent the buildings the players chose to produce in real time, and itemsets denote time windows of 30 seconds. We consider only the 10 first minutes of each game: after that, buildings importance fade away and do not translate anymore into strategy blocks.

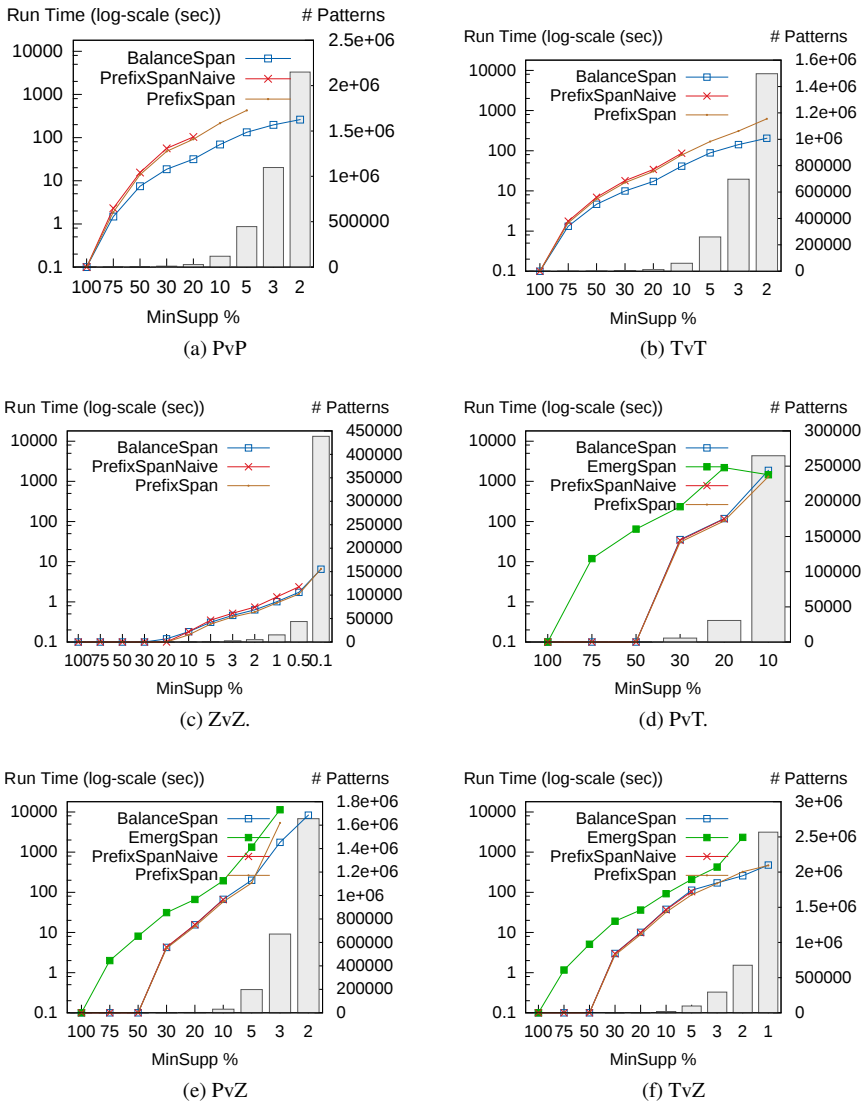
7.2 Experimental results

We report an empirical evaluation of our algorithms using real datasets. Experiments are performed on a 1.8 GHz Intel Core i5 with 8 GB main memory running Mac OS X 10.9.1. We started from the original C++ version of PREFIXSPAN [15] to implement the algorithms EMERGSPAN, PREFIXSPANNAIVE and BALANCESPAN (compiled with g++ and -03 optimization). The source codes and the data sets are available[5].

[3] http://en.wikipedia.org/wiki/Real-time_strategy

[4] http://wiki.teamliquid.net/starcraft2/Replay_Websites

[5] `http://liris.cnrs.fr/mehdi.kaytoue/balancespan.html`



(a) PvP (b) TvT

(c) ZvZ. (d) PvT.

(e) PvZ (f) TvZ

Figure 1: Run time and number of patterns.

The results are analyzed following different criteria: the runtime aspect, the memory usage and the actionability of the discovered patterns.

Runtime. We discuss running times of the proposed algorithms. Firstly, we consider the non-mirror databases given by transformed interaction databases $\mathcal{D}_t$ and signed databases $\mathcal{D}_s$. For different minimum frequency thresholds σ, we present the runtime of PREFIXSPAN on $\mathcal{D}_s$ as a rough baseline (since it does not compute the balance of a pattern), and the runtimes of the three others algorithms on their respective data representation. Limited by space, we report the results for the datasets PvZ and TvZ only (omitting PvT). It follows that BALANCESPAN, our general algorithm, is the only one able to be executed with lowest σ (Figure 1 (c) and (d)), since it produces a lot of useless nodes (see Figure 2 and the next paragraph). We report the same results for mirror

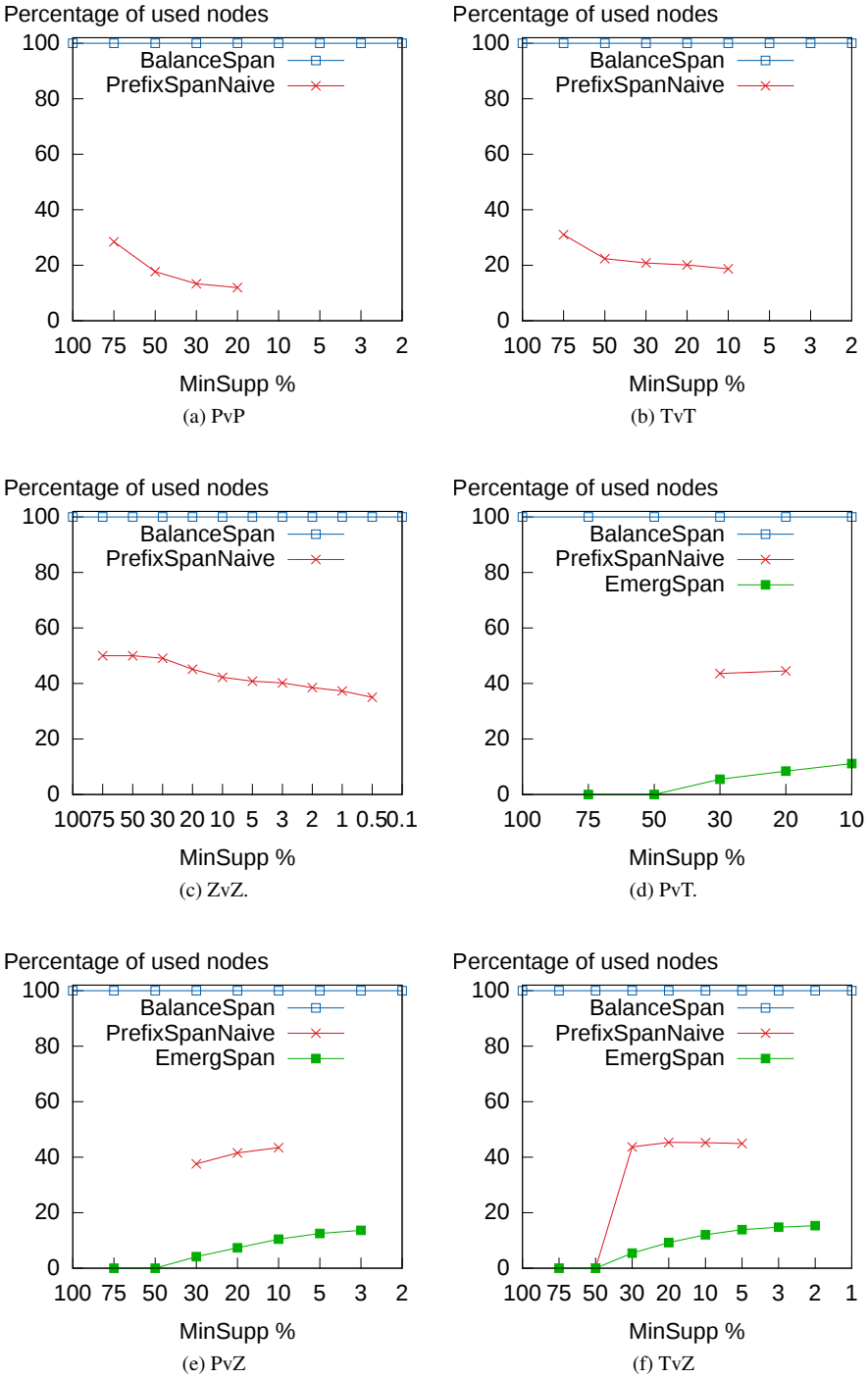


Figure 2: Percentage of used nodes.

Table 5: Datasets: sequence and item counts; max. and avg. sequence sizes (s_{max}, s_{avg}); max. and avg. itemsets (i_{max}, i_{avg}).

Dataset	$\|\mathcal{D}\|$	$\|\mathcal{I}\|$	s_{max}	s_{avg}	i_{max}	i_{avg}
$\mathcal{D}_s$ - PvP	6,823	26	42	21.7	6	1.8
$\mathcal{D}_s$ - PvT	19,270	62	42	25.9	8	1.8
$\mathcal{D}_s$ - PvZ	23,491	52	41	23.1	7	1.7
$\mathcal{D}_s$ - TvT	7,598	36	38	23.8	7	1.8
$\mathcal{D}_s$ - TvZ	24,459	62	37	21.2	8	1.6
$\mathcal{D}_s$ - ZvZ	9,922	26	28	10.4	7	1.4
$\mathcal{D}_t$ - PvT	19,270	34	43	26.9	8	1.8
$\mathcal{D}_t$ - PvZ	23,491	29	42	24.1	7	1.7
$\mathcal{D}_t$ - TvZ	24,459	34	38	22.2	8	1.6

datasets (i.e. $\mathcal{D}_s$ only for PvP, TvT and ZvZ) in Figure 1 (a) and (b). BALANCESPAN clearly outperforms PREFIXSPANNAIVE, its only concurrent (remembering that PREFIXSPAN is given as baseline since it does not compute the balances, and EMERGSPAN does not apply for mirror databases. Note that on the figure, missing points correspond to unterminated runs where available memory is insufficient. It is also interesting to note that runtimes are linked to the percentage of used nodes in the structure of tree generated by the algorithms (Figure 2) which we explain now: indeed, the more the percentage of used nodes is, the more comparable to BALANCESPAN the algorithms are in term of runtime.

Memory usage. Each of the proposed algorithms builds a pattern tree in which each node represents a frequent sequential pattern, but not necessarily a frequent balanced pattern from $\mathcal{F}_t$ (or $\mathcal{F}_s$). In Figure 2, we report the percentage of tree nodes that effectively produce frequent balanced patterns. BALANCESPAN always produces useful nodes, whereas the other algorithms produce always more than 50% of useless nodes. This explains why BALANCESPAN can deal with lower minimum frequency thresholds and is faster than the others. Another important parameter is the scalability of the algorithms. Some algorithms are not able to compute the output patterns for a minimal frequency support too low because they require too much memory to run. For example, the PREFIXSPANNAIVE and EMERGSPAN algorithms can not reach a minimal frequency threshold lower than 0.1 in most cases. However BALANCESPAN is able to output frequent patterns with a minimal frequency threshold closed to 0.01. Note that the original PREFIXSPAN algorithm may process lower minimal frequency threshold since it does not compute the support of the dual pattern and thus the balance measure which requires more memory usage.

Discovered patterns. It is interesting to visualize the distribution of both the support and the balance of the patterns. Figure 3 gives such distribution for dataset ZvZ that allows very fast computations with low σ (less than 5 seconds for $\sigma = 0.001$). There, both a pattern and its dual are presented, which allows interestingly to observe that $y = 0.5$ gives almost a symmetry axis. Indeed, both a pattern and its dual do not necessarily have the same support. On can notice that empirically, there are high chances for a pattern with high frequency to have a fair balance around 0.5.

We observed the collection of resulting patterns with a game expert. Firstly, the distribution Support/Balance (Figure 3) is an expected result: the game in its current state is globally balanced. Empirically again, the less frequent a pattern the more chances it is a long sequence (limited by space we do not present pattern size distribution: in average the size is 10 with standard deviation of 3.2). This makes also sense, since the

goal of any competitive strategy game is that at the initial state of the game balance should be exactly 0.5 and the longer the game the easiest it is to break the symmetry for any of the two players depending of their skills.

There are $43,610$ pattern on for ZvZ with $\sigma = 0.001$. We filtered out the pattern with only one player (containing only either $+$ or $-$) which returns $40,674$ patterns. Then we restricted the set of pattern to those involving two specific items (RoachWarren and Spire) to get only 290 patterns. $\langle SpawPool^{+}, SpawnPool^{-}, SpiCrawler^{+}, RoachWarren^{+}, Spire^{-}\rangle$ denotes for example games where one of the player go to air units and the second to ground units, two different known openings, with balance 0.47 and support 68. Clearly, the resulting set of patterns of any extraction is not human readable. However, it can be queried by a live program during a game, which is a very demanded tool for broadcasters of E-Sport competitions to better interact with their audience [5, 17, 11]. This is one of our objective to include such a possibility in Starcraft II Observer modules[6].

8 Conclusion

In this work, we present efficient algorithms and show how emerging pattern mining is used to analyze video game strategy balance issues. It is our firm belief that our preliminary investigation on sequential patterns from zero-sum games will provide valuable results, but will require significant further research, for instance using more theoretical foundations from game theory to build predictive tools. We hope that this work will motivate and encourage research projects in this new exciting and challenging area.

References

[1] Arnaud Giacometti, Dominique H. Li, Patrick Marcel, Arnaud Soulet. 20 Years of Pattern Mining: a Bibliometric Survey. *SIGKDD Explorations*, 15(1):41–50, 2014.

[2] Stephen D. Bay and Michael J. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5(3):213–246, 2001.

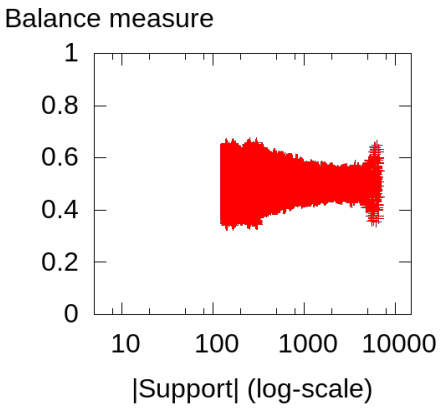
[3] Guillaume Bosc, Mehdi Kaytoue, Chedy Raïssi, Jean-François Boulicaut, and Philip Tan. Mining balanced patterns in real-time strategy games. In *European Conference on Artificial Intelligence (ECAI'14)*, 2014.

[4] Sarah Chan, Ben Kao, Chi Lap Yip, and Michael Tang. Mining emerging substrings. In *DASFAA*, pages 119–126. IEEE Computer Society, 2003.

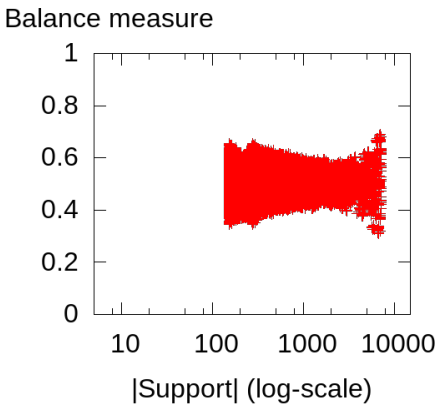
[5] Gifford Cheung and Jeff Huang. Starcraft from the stands: understanding the game spectator. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 763–772, New York, NY, USA, 2011. ACM.

[6] Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: discovering trends and differences. In *ACM SIGKDD*, KDD '99, pages 43–52, New York, NY, USA, 1999. ACM.

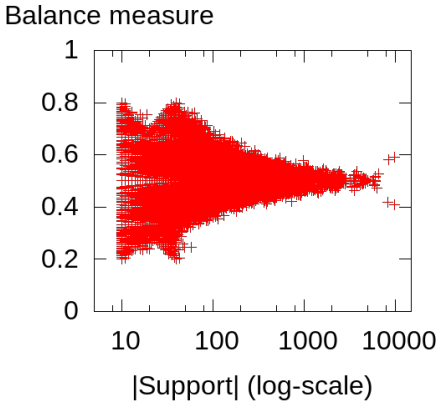
[6] http://gamelab.mit.edu/mit-overseer-observer-mod/



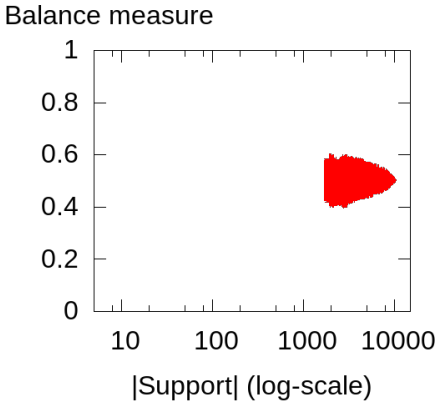
(a) PvP - $\sigma = 0.02$.



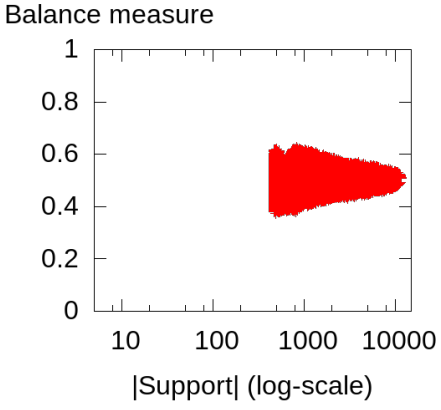
(b) TvT - $\sigma = 0.02$.



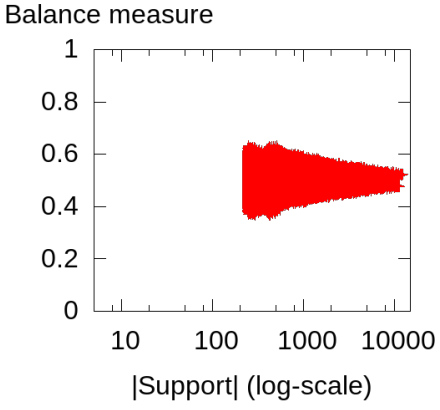
(c) ZvT - $\sigma = 0.001$.



(d) PvT - $\sigma = 0.1$.



(e) PvZ - $\sigma = 0.1$.



(f) TvZ - $\sigma = 0.01$.

Figure 3: Balance vs. Support for dataset ZvZ and $\sigma = 0.001$.

[7] Hongjian Fan and Kotagiri Ramamohanarao. Fast discovery and the generalization of strong jumping emerging patterns for building compact and accurate classifiers. *IEEE Trans. Knowl. Data Eng.*, 18(6):721–737, 2006.

[8] Pedro González Franciso Herrera, Cristóbal José Carmona and María José del Jesus. An overview on subgroup discovery: foundations and applications. *Knowledge and Information Systems*, 29(3):495–525, 2011.

[9] Milton García-Borroto, JoséFco. Martínez-Trinidad, and JesúsAriel Carrasco-Ochoa. A survey of emerging patterns for supervised classification. *Artificial Intelligence Review*, pages 1–17, 2012.

[10] Wei Gong, Feida Zhu, Alakorn Achananuparp, and David Lo. On finding the point where there is no return: Turning point mining on game data. In *SIAM Data Mining Conference*, 2014.

[11] Mehdi Kaytoue, Arlei Silva, Loïc Cerf, Wagner Meira Jr., and Chedy Raïssi. Watch me playing, i am a professional: a first study on video game live streaming. In *Proc of WWW'12 (Companion Vol.)*, pages 1181–1188. ACM, 2012.

[12] Sergei O. Kuznetsov. Galois connections in data analysis: Contributions from the soviet era and modern russian research. In *Formal Concept Analysis*, volume 3626 of *LNCS*, pages 196–225. Springer, 2005.

[13] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[14] Petra K. Novak, Nada Lavrač, and Geoffrey I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *J. Mach. Learn. Res.*, 10:377–403, 2009.

[15] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *ICDE*, pages 215–224. IEEE Computer Society, 2001.

[16] Marc Plantevit and Bruno Crémilleux. Condensed representation of sequential patterns according to frequency-based measures. In *Advances in Intelligent Data Analysis*, volume 5772 of *LNCS*, pages 155–166. Springer, 2009.

[17] T. L. Taylor. *Raising the Stakes : E-Sports and the Professionalization of Computer Gaming*. MIT Press, 2012.

[18] Xifeng Yan, Jiawei Han, and Ramin Afshar. Clospan: Mining closed sequential patterns in large databases. In Daniel Barbará and Chandrika Kamath, editors, *SDM*. SIAM, 2003.

Contents