A Pattern Mining Approach to Study Strategy Balance in RTS Games

Guillaume Bosc, Philip Tan, Jean-François Boulicaut, Chedy Raïssi, Mehdi Kaytoue

A Pattern Mining Approach to Study Strategy Balance in RTS Games

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

[1]Université de Lyon, CNRS, INSA-Lyon, LIRIS, UMR5205, F-69621, France
[2]MIT Game Lab, Cambridge (MA), USA
[3]LORIA (CNRS - Inria NGE - Université de Lorraine), Vandoeuvre-lès-Nancy, France

***Abstract*—Whereas purest strategic games such as Go and Chess seem timeless, the lifetime of a video game is short, influenced by popular culture, trends, boredom and technological innovations. Even the important budget and developments allocated by editors cannot guarantee a timeless success. Instead, novelties and corrections are proposed to extend an inevitably bounded lifetime. Novelties can unexpectedly break the balance of a game, as players can discover unbalanced strategies that developers did not take into account. In the new context of electronic sports, an important challenge is to be able to detect game balance issues. In this article, we consider real time strategy games (RTS) and present an efficient pattern mining algorithm as a basic tool for game balance designers that enables one to search for unbalanced strategies in historical data through a Knowledge Discovery in Databases process (KDD). We experiment with our algorithm on StarCraft II historical data, played professionally as an electronic sport.**

***Index Terms*—Mining methods and analysis, video game**

I. Introduction

The recent and fast development of the video game industry has been catalyzed by technological innovations, a democratized access to connected electronic devices, new economic models (*free* games where users may pay for extra contents), and recently with competitive gaming (esports) and video game live streaming platforms [1]. People not only enjoy playing, but also enjoy learning from watching others performing, as a daily leisure activity [2], [3].

Producing a video game is an expensive process, thus, keys to a massive, immediate and durable success are sought. Pragmatically however, one attempts to extend the game lifetime after the release, by correcting bugs, introducing new features and considering user feedbacks. Whereas it could be easily argued that bugs are not acceptable after a release, it is hard to predict the results of human creativity in presence of rich environments that are video games.

Hopefully, companies realized in the current *big data* atmosphere that the tremendous amounts of game behavioral data they store are valuable to face many new challenges such as: detecting unexpected situations and bugs [4] cheaters [5], designing artificial agents [6], improving match making systems and adjusting game difficulty [7]. Analyzing these massive sets of historical data by means of visualization, machine learning and data mining techniques is at the heart of *video game analytics* for enhancing user experience and extending game lifetime [4].

This context roots the motivation of our work: behavioral data can help to study the balance of a game, that is, to adjust the rules over time while still enabling novel rules to counter boredom. This is especially important for games played as an *electronic sport*, and also for game lifetime extension in general. We will focus on the concept of *balance* which is a core concept in competitive game design: it consists of defining and tuning the basic rules that prevent extreme situations, thus balancing fairness and competitive aspects.

In this article, we define and mine *patterns* in game historical data for a better understanding of balance issues in RTS games. The intuition of the *balanced sequential pattern discovery* problem is the following. Consider a set of games, each of them represented by a sequence of actions of two players (thus entailing the player strategy). The problem is to find patterns as sequence generalizations that frequently occur in the historical data and whose balance is given by proportions of their wins and losses. Our goal is twofold: (i) we give the basic algorithmic tools that enable an efficient pattern mining, (ii) we show that the extracted patterns reveal interesting knowledge:

- (i) We revisit the problem of strategy elicitation from two player RTS games by differentiating two cases: when both players have access to (a) different game actions and (b) the same game actions. In the first case, we show how existing pattern mining methods enable with slight modifications to discover frequent strategies and compute a balance measure. In the second case, the most general, existing approaches fail: we propose an original algorithm, BalanceSpan.
- (ii) We show through experiments that BalanceSpan is scalable and able to discover patterns of interest in a large StarCraft II dataset that can help detecting balance issues. For that matter, we anchor our algorithm in a Knowledge Discovery in Databases process [8]. Pattern mining is one of the many steps of this interactive and iterative process guided by an expert of the data domain who selects and interprets the patterns [9], [10].

The paper is organized as follows. Section II recalls the basics of sequential pattern mining before the introduction of our mining problem in Section III. Our method is developed in sections IV and V. Algorithms are designed (Section VI) and experimented with E-Sport data (Section VII) before overviewing related work and concluding.

II. Preliminaries

We recall the basic definitions of frequent sequential patterns [11] and emerging patterns [12] useful in the sequel. 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 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 indicates 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 I (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 $X_1 \subseteq X'_{j_1}, X_2 \subseteq X'_{j_2}, ..., X_{l_s} \subseteq X'_{j_{l_s}}$.

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

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

In some cases, 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 into two databases, 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 [12], [13].

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

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

Example. Let $\mathcal{D} = \{s_1, s_2, s_3, s_4\}$ with $\mathcal{I} = \{a,\ b,\ c,\ d,\ e,\ f,\ g\}$ be the sequence database given in Table I. For brevity, we drop commas and 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. We have $growth\_rate(\langle cb \rangle, \mathcal{D}^-, \mathcal{D}^+) = \frac{2}{2} \times \frac{2}{1} = 2$, i.e., $\langle cb \rangle$ is twice more present in class $-$ than in class $+$.

TABLE I
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	$\langle eg\{af\}cbc \rangle$	$-$

III. The Problem of Strategy Elicitation

A zero-sum game, or competitive interaction, can be modeled as a sequence of actions performed by two players where exactly one player wins (no ties). Such a sequential game can be represented as a sequence of sets of actions, each action performed 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 $\mathcal{R}$ is called an *interaction database*. Each sequence denotes one single game, i.e., an interaction between two players, and is defined over the set of items $\mathcal{I} = Actions \times Players$. A mapping $class : \mathcal{R} \to Players$ gives the winner of each interaction.

Example. In Table II, 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 d while p_2 did a. At the end, the player p_1 wins*".

Given an interaction database, the problem 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 a win or loss through a so-called *balance* measure. In the current sequential pattern mining settings, the goal is to find frequent sub-sequences of actions (i.e., 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 loser 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 II) does not fulfill our objectives: we wish to discriminate victories and not victorious players themselves. As such, existing emerging sequential pattern mining methods and algorithms cannot be used to answer our problem.

We propose to differentiate two cases of interaction 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 (Section II) are able to answer the problem by slightly modifying the interaction database representation. In the second case, the most general one, 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 IV. This is formalized in the two next sections, and it enables the design of efficient pattern mining algorithms in Section VI.

TABLE II
An interaction sequence database $\mathcal{R}$

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

IV. Balanced Patterns in Non-mirror Databases

In this section, we consider an interaction database, called a *non-mirror database*, where the set of actions is different for each of the players in a single interaction. It means that we only have two types of players in each sequence and in the whole database (e.g., Protoss and Zerg factions in the RTS game StarCraft II), and these types are determined by the actions they can do. As such, the type can also be used as a winning class label. To characterize balanced patterns in such databases, we consider a simple transformation of the original interaction database $\mathcal{R}$ by dropping the player associated to each action, and labeling each sequence by the type of the winner. This enables 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 $\mathcal{T}$ 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 $class : \mathcal{T} \rightarrow \{\mathcal{I}_1, \mathcal{I}_2\}$ is called a *transformed interaction database*.

Consider an interaction $s \in \mathcal{T}$ 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 the emerging patterns settings. Indeed, consider an arbitrary pattern s over $\mathcal{I}_1 \cup \mathcal{I}_2$: its support in the whole database $sup(s,\mathcal{T})$ tells us its frequency, while $sup(s,\mathcal{T}^{\mathcal{I}_1})$ and $sup(s,\mathcal{T}^{\mathcal{I}_2})$ enable to define a balance measure as a growth-rate.

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

$$balance(s, \mathcal{T}^k) = \frac{|sup(s, \mathcal{T}^k)|}{|sup(s, \mathcal{T}^1)| + |sup(s, \mathcal{T}^2)|}$$

Remark. The balance measure is a normalized version of the growth rate given in previous section such that $balance(.) \in]0, 1]$ and $balance(s, \mathcal{T}^1) + balance(s, \mathcal{T}^2) = 1$ which entails a zero-sum game property.

Example. Table III gives a transformed interaction database $\mathcal{T}$, obtained from a non-mirror interaction database $\mathcal{R}$, with $\mathcal{I}_1 = \{a, b\}$ and $\mathcal{I}_2 = \{c\}$ being the sets of actions of each player type. With $\sigma = 0.2$, $s = \langle\{ab\}\{c\}\rangle$ is a frequent balanced pattern since $freq(s, \mathcal{T}^{\mathcal{I}_1}) = \frac{2}{3}$ and $freq(s, \mathcal{T}^{\mathcal{I}_2}) = \frac{1}{2}$. Moreover, $balance(s, \mathcal{T}^{\mathcal{I}_1}) = \frac{2}{3}$ and $balance(s, \mathcal{T}^{\mathcal{I}_2}) = \frac{1}{3}$. It means that s wins two times more for the type 1 of player than for the type 2.

TABLE III
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 IV
A signed interaction database.

id	$s \in \mathcal{S}$
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	$\langle a^-\{b^+c^+\}\rangle$

V. 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 can not apply. We propose a new interaction database representation, *signed interaction databases*. It enables 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 $\mathcal{S}$ is built from an interaction database $\mathcal{R}$ as follows: Each action of an interaction sequence is signed $+$ if it is performed by the winner and signed $-$ if performed by the loser (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 IV, we have $\mathcal{I}_s = \{a, b, c, d\} \times \{+, -\}$, $dual(a^+) = a^-$ and $dual(s_1) = \langle a^-\{b^-c^+\}\rangle$.

These definitions enable 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 $\mathcal{S}$. The balance measure of s is

$$balance(s) = \frac{|sup(s,\mathcal{S})|}{|sup(s,\mathcal{S})| + |sup(dual(s),\mathcal{S})|} \quad (1)$$

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 $s \in \mathcal{S}$. Consider the following example: $\mathcal{S} = \{\langle\{a^+b^-\}\{a^-b^+\}\rangle, \langle\{a^-b^+\}\rangle\}$ with $\sigma = \frac{1}{2}$. The sequence $s = \langle\{a^+b^-\}\rangle$ is a frequent sequential pattern, and $|sup(s,\mathcal{S})| = 1$. We have also $|sup(dual(s),\mathcal{S})| = |sup(dual(\langle\{a^-b^+\}\rangle),\mathcal{S})| = 2$. 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 in which we ignore sequences where both a pattern and its dual appear.

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

$$balance_{gen}(s) = \frac{|sup(s,\mathcal{S}) \backslash sup(dual(s),\mathcal{S})|}{|sup(s,\mathcal{S}) \sqcup sup(dual(s),\mathcal{S})|} \quad (2)$$

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.

Problem (Mining balanced patterns from signed interactions). Let $\mathcal{S}$ be a signed interaction database defined over $\mathcal{I}_s$ generated from an interaction database $\mathcal{R}$, 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 $s \in \mathcal{F}_s$, $freq(s,\mathcal{S}) \geq \sigma$, $freq(dual(s),\mathcal{S}) \geq \sigma$ and the balance measure is computed and given by (2). Furthermore, the fact that 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 $common(s) = |sup(s,\mathcal{S}) \cap sup(dual(s,\mathcal{S}))|$ 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 IV gives a signed interaction database $\mathcal{S}$ obtained from an arbitrary $\mathcal{R}$. With $\sigma = \frac{1}{4}$, $s = \langle a^+c^-\rangle$ appears two times, its dual appears only once, hence $balance(s) = \frac{2}{3}$.

Remark. Any interaction database, mirror or non-mirror, can be represented as a signed interaction database. For the non-mirror case, one can easily prove that for any balanced pattern s, we have $common(s) = \emptyset$ and thus Formula (1) holds.

VI. Algorithms

We present several 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 [11].

A. 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 [11]. Firstly, the database $\mathcal{D}$ is scanned once 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 prefixes sequences 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 generate 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 $\lceil \sigma \times |\mathcal{D}| \rceil$ 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.

Example. We briefly illustrate PrefixSpan on the toy example of the Table I with $\sigma = 0.5$. A larger example is available in the original publication of PrefixSpan [11]. The first step of PrefixSpan consists of finding frequent item from $\mathcal{D}$: $\langle a\rangle$ ($|sup(\langle a\rangle,\mathcal{D})| = 4$), $\langle b\rangle$ ($|sup(\langle b\rangle,\mathcal{D})| = 4$), $\langle c\rangle$ ($|sup(\langle c\rangle,\mathcal{D})| = 4$), $\langle d\rangle$ ($|sup(\langle d\rangle,\mathcal{D})| = 3$), $\langle e\rangle$ ($|sup(\langle e\rangle,\mathcal{D})| = 3$) and $\langle f\rangle$ ($|sup(\langle f\rangle,\mathcal{D})| = 3$). For each of these previous frequent sequential patterns s of size 1, PrefixSpan projects $\mathcal{D}$ into a projected database with prefix s. Thus, the $\langle a\rangle$-projected database $\mathcal{D}_{|\langle a\rangle}$ is composed of 4 sequences: $\langle\{abc\}\{ac\}d\{cf\}\rangle$, $\langle\{\_d\}c\{bc\}\{ae\}\rangle$, $\langle\{\_b\}\{df\}cb\rangle$ and $\langle\{\_f\}cbc\rangle$. Then, PrefixSpan searches for frequent sequential patterns of size 2: $\langle aa\rangle$ ($|sup(\langle aa\rangle,\mathcal{D}_{|\langle a\rangle})| = 2$), $\langle ab\rangle$ ($|sup(\langle ab\rangle,\mathcal{D}_{|\langle a\rangle})| = 4$), $\langle(ab)\rangle$ ($|sup(\langle(ab)\rangle,\mathcal{D}_{|\langle a\rangle})| = 2$), $\langle ac\rangle$ ($|sup(\langle ac\rangle,\mathcal{D}_{|\langle a\rangle})| = 4$), $\langle ad\rangle$ ($|sup(\langle ad\rangle,\mathcal{D}_{|\langle a\rangle})| = 2$) and $\langle af\rangle$ ($|sup(\langle af\rangle,\mathcal{D}_{|\langle a\rangle})| = 2$). Then for each of these sequential patterns of size 2 PrefixSpan creates the projected databases and extracts frequent patterns of size 3, and so on.

B. 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 $\mathcal{R}$ as a transformed database $\mathcal{T}$ 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 and can be computed in negligible time as a post-processing [14]. Based on PrefixSpan, EmergSpan works as follows. For each sequence of the database $\mathcal{T}$, its class is appended

ALGORITHM: PREFIXSPAN

Require: A sequence s, the s-projected DB $\mathcal{D}_{|s}$, and a threshold σ
Ensure: The frequent sequential pattern set F
1: $F \longleftarrow \{s\}$;
2: scan $\mathcal{D}_{|s}$ once, find every frequent item a such that
(a) s can be extended to $(s \circ_i a)$, or
(b) s can be extended to $(s \circ_s a)$
3: **if** no valid a available **then**
4: **return** F;
5: **end if**
6: **for all** valid a **do**
7: (a) $F \longleftarrow F \cup PrefixSpan(s \circ_i a, \mathcal{D}_{|s \circ_i a}, \sigma)$ or
8: (b) $F \longleftarrow F \cup PrefixSpan(s \circ_s a, \mathcal{D}_{|s \circ_s a}, \sigma)$
9: **end for**
10: **return** F;

ALGORITHM: EMERGSPAN

Require: A sequence s, the s-projected transformed DB $\mathcal{T}_{|s}$, and a minimum threshold σ
Ensure: The frequent sequential pattern set F
1: $F \longleftarrow \emptyset$;
2: **if** s ends with the class **then**
3: Compute $balance(s)$ thanks to $sup(parent(s))$
4: $F \longleftarrow \{s\}$
5: **return** F;
6: **end if**
7: scan $\mathcal{T}_{|s}$ once, find every frequent item a such that
(a) s can be extended to $(s \circ_i a)$, or
(b) s can be extended to $(s \circ_s a)$
8: **if** no valid a available **then**
9: **return** F;
10: **end if**
11: **for all** valid a **do**
12: (a) $F \longleftarrow F \cup EmergSpan(s \circ_i a, \mathcal{T}_{|s \circ_i a}, \sigma)$ or
13: (b) $F \longleftarrow F \cup EmergSpan(s \circ_s a, \mathcal{T}_{|s \circ_s a}, \sigma)$
14: **end for**
15: **return** F;

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, allowing a direct computation of the balance measure.

Example. To illustrate how EMERGSPAN works, let us consider the previous example on the transformed interaction database $\mathcal{T}$ of Table III. Remind that this database is obtained from a non-mirror interaction database $\mathcal{R}$, with $\mathcal{I}_1 = \{a, b\}$ and $\mathcal{I}_2 = \{c\}$ being the sets of actions of each player type. Thus, we transformed each of these sequences by appending the item $\mathcal{I}_1$ (resp. $\mathcal{I}_2$) at the end if the player with actions in $\mathcal{I}_1$ (resp. $\mathcal{I}_2$) won. For example, s_2 becomes $\langle\{ab\}ac\mathcal{I}_1\rangle$. The exploration of EMERGSPAN works the same way as PREFIXSPAN excepted that it only keeps patterns that end up with either $\mathcal{I}_1$ or $\mathcal{I}_2$. Thus, computing balance of pattern $\langle X_1 \dots X_l \mathcal{I}_j\rangle$ only requires to backtrack to the parent (i.e., $\langle X_1 \dots X_l\rangle$) to get its support to evaluate the balance. So, when EMERGSPAN deals with the sequential pattern $s = \langle\{ab\}c\mathcal{I}_1\rangle$, since the last item of s is $\mathcal{I}_1$, it will output this pattern, and to compute its balance measure, it uses the support of its parent node $s' = \langle\{ab\}c\rangle$: $balance(s) = \frac{|sup(s,\mathcal{T})|}{|sup(s',\mathcal{T})|} = 0.67$.

C. Mining Balanced Patterns with PREFIXSPANNAIVE

Now we consider a signed interaction database $\mathcal{S}$ over $\mathcal{I}_s$ and a minimal frequency threshold σ. The problem is to extract the set of frequent sequential patterns and compute their balanced measure. We 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 outputted. 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 an 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 outputted, $dual(q)$ is flagged as already outputted (redundant pattern) and the process recursively continues. Otherwise, the algorithm backtracks. In this way, q and $dual(q)$ are never outputted together. 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 direct: 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.

Example. PREFIXSPANNAIVE requires a signed intereaction database. Let us consider that of Table IV with $\sigma = 1/4$ (i.e., frequent items are items that only appear once). PREFIXSPANNAIVE starts with an empty sequence $s = \langle\rangle$ ($dual(\langle\rangle) = \langle\rangle$) and the entire database. First, it searches for frequent items: e.g., both a^+ and a^- are frequent. Thus, PREFIXSPANNAIVE flags that $\langle a^-\rangle$ is the dual of $\langle a^+\rangle$, and calls $PrefixSpanNaive(\langle a^+\rangle, \mathcal{S}_{|\langle a^+\rangle}, \sigma)$. Then, PREFIXSPANNAIVE searches for frequent items in $\mathcal{S}_{|\langle a^+\rangle}$: e.g., c^- is frequent. So, it will search in the children of $\langle a^-\rangle$ if there is c^+, but at this step, the node of $\langle a^-\rangle$ has not been explored yet. Thus, when $\langle a^-\rangle$ is explored, PREFIXSPANNAIVE extracts frequent items in $\mathcal{S}_{|\langle a^-\rangle}$: e.g., c^+ is frequent. So it will search among the children of $dual(\langle a^-\rangle)$ if $dual(c^+)$ exists: $\langle a^- c^+\rangle$ is a frequent sequential pattern that could be outputted. Finally, PREFIXSPANNAIVE will proceed iteratively until none sequential patterns could be extended.

D. Mining Balanced Patterns with BALANCESPAN

The problem of PREFIXSPANNAIVE 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 VII), 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

ALGORITHM: PREFIXSPANNAIVE

Require: A sequence s, the s-projected signed DB $\mathcal{S}_{|s}$, and a minimum threshold σ
Ensure: The frequent sequential pattern set F
Compute $balance(s)$ thanks to $dual(s)$
$F \longleftarrow \{s\}$;
scan $\mathcal{S}_{|s}$ once, find every frequent item a such that
  (a) s can be extended to $(s \circ_i a)$, or
  (b) s can be extended to $(s \circ_s a)$
if no valid a available **then**
  return F;
end if
for all valid a **do**
  (a) Search for $dual(s \circ_i a)$ among children of $dual(s)$
  if $dual(s \circ_i a)$ exists **then**
    Link $dual(s \circ_i a)$ to $s \circ_i a$
    $F \longleftarrow F \cup PrefixSpanNaive(s \circ_i a, \mathcal{S}_{|s \circ_i a}, \sigma)$
  end if
  or,
  (b) Search for $dual(s \circ_s a)$ among children of $dual(s)$
  if $dual(s \circ_s a)$ exists **then**
    Link $dual(s \circ_s a)$ to $s \circ_s a$
    $F \longleftarrow F \cup PrefixSpanNaive(s \circ_i a, \mathcal{S}_{|s \circ_s a}, \sigma)$
  end if
end for
return F;

ALGORITHM: BALANCESPAN

Require: A sequence s, its dual sequence $s' = dual(s)$, the s-projected signed DB $\mathcal{S}_{|s}$, the s'-projected signed DB $\mathcal{S}_{|s'}$, and a minimum threshold σ
Ensure: The frequent balanced pattern set F
Compute $balance(s)$
$F \longleftarrow \{(s, s')\}$;
scan $\mathcal{S}_{|s}$ once, find every frequent item a such that
  (a) s can be extended to $(s \circ_i a)$, or
  (b) s can be extended to $(s \circ_s a)$
scan $\mathcal{S}_{|s'}$ once, find every frequent item b such that
  (a) s' can be extended to $(s' \circ_i b)$, or
  (b) s' can be extended to $(s' \circ_s b)$
if no valid a or b available **then**
  return F;
end if
for all valid a and b such that $a = dual(b)$ **do**
  (a) $F \longleftarrow F \cup BalanceSpan(s \circ_i a, s' \circ_i b, \mathcal{S}_{|s \circ_i a}, \mathcal{S}_{|s' \circ_i b}, \sigma)$
  (b) $F \longleftarrow F \cup BalanceSpan(s \circ_s a, s' \circ_s b, \mathcal{S}_{|s \circ_s a}, \mathcal{S}_{|s' \circ_s b}, \sigma)$
end for
return F;

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 $\mathcal{S}_{|s \circ a}$ and consequently a new node in the pattern tree, we consider simultaneously an item and its dual, hence two projected databases $\mathcal{S}_{|s \circ a}$ and $\mathcal{S}_{|dual(s) \circ dual(a)}$ are related to a single node (this is done for both kinds of extensions $\circ_i$ and $\circ_s$). Thus, it ensures 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.

Example. BALANCESPAN also requires a signed interaction database, but contrary to PREFIXSPANNAIVE, it proceeds to a double projection at a time. Let us still consider the toy dataset of Table IV with $\sigma = 1/4$. Starting with the empty sequence and the entire dataset, the first step consists of finding frequent items: e.g., a^+ is frequent. Contrary to PREFIXSPANNAIVE, BALANCESPAN directly generates the dual of $\langle a^+ \rangle$ to proceed to the double projection. Thus, it checks if a^- is frequent and then projects on both $\langle a^+ \rangle$ and $\langle a^- \rangle$ at a time: i.e., it creates a new node in the pattern tree that is related to the couple $(\langle a^+ \rangle, \langle a^- \rangle)$. The next step calls $BalanceSpan$ on this new node. So it searches for frequent items in the projected database $\mathcal{S}_{|\langle a^+ \rangle}$: e.g., c^-. Thus it checks if $dual(c^-)$ is frequent in $\mathcal{S}_{|\langle a^- \rangle}$: the node containing the couple $(\langle a^+ c^- \rangle, \langle a^- c^+ \rangle)$ is created and explored in the next step.

VII. EXPERIMENTS

A. StarCraft II in a nutshell

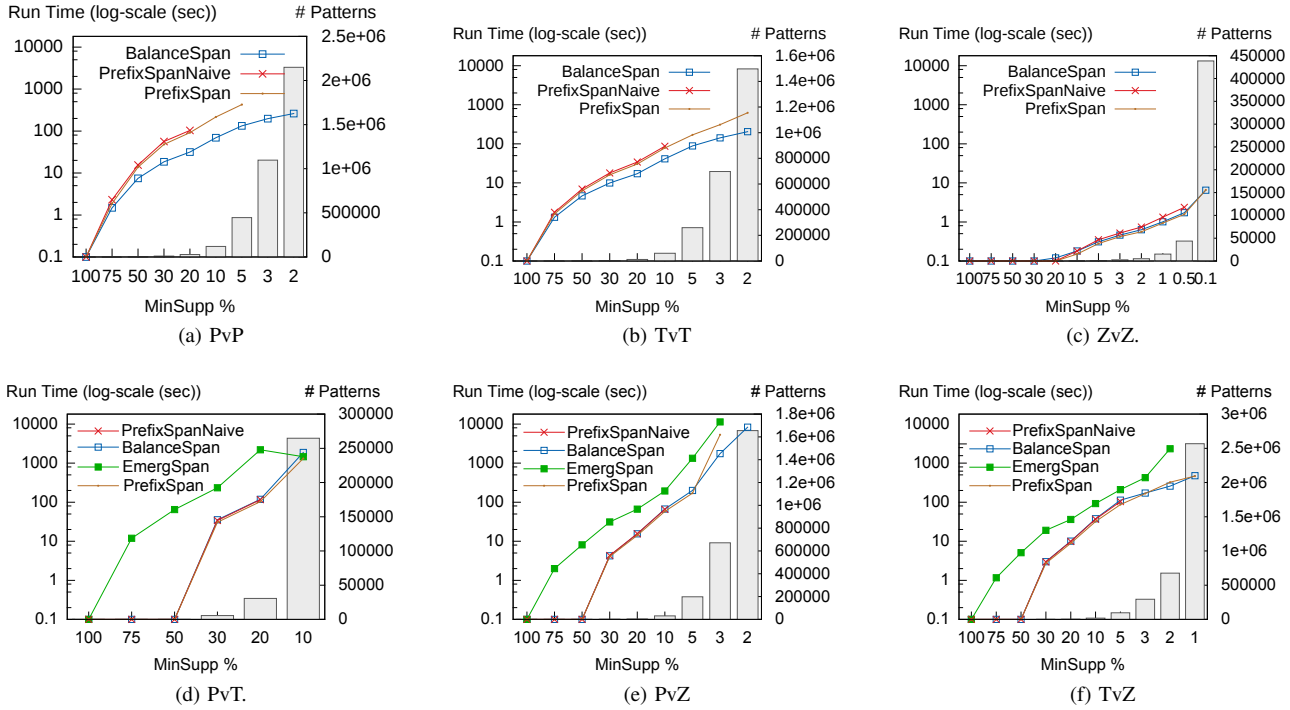
We study one of the most competitive real-time strategy games (RTS), StarCraft II (Blizzard Entertainment, 2010), successor of *StarCraft: Brood War*, test bed for many research in AI [6]. A game involves two players each choosing a faction among Zerg (Z), Protoss (P) and Terran (T): there are 6 different 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. 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. As such, there are mirror match-ups (TvsT, PvsP, ZvsZ) and non-mirror match-ups (TvsP, TvsZ, PvsZ). A strategy is hidden in large sequences of actions generated by players and called *replays*.

Played as an electronic sport, StarCraft II is regularly patched: basic rules of the games are adjusted (properties of units, building times, ...), new rules are introduced through expansion sets (*heart of the swarm* and *legacy of the void*). The balance design team of StarCraft II, often needs to study historical data, care about player feedback on Web forums, and finally to justify their choices. After quantitative experiments of our algorithms, we will discuss the usefulness of our approach to help studying balance issues in RTS games.

B. Datasets

StarCraft II replays are files that store any action performed by all players during a game, and are easily accessible on the Web [1]. 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. A game is selected if it involves a high level players (in the highest leagues and playing at least 200 actions per minute), 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 up. Buildings are one of the key elements of a strategy, since they enable different kinds of units production: from

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



(a) PvP (b) TvT (c) ZvZ.

(d) PvT. (e) PvZ (f) TvZ

Fig. 1. Run time and number of patterns.

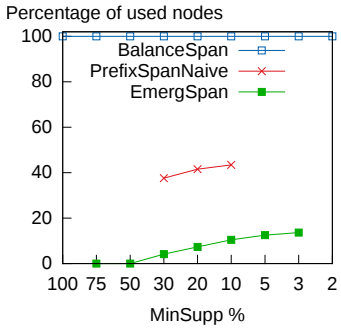


Fig. 2. Percentage of used nodes in the PvZ dataset.

TABLE V
DATASETS: SEQUENCE AND ITEM COUNTS; MAX. AND AVG. SEQUENCE SIZES (s_{max}, s_{avg}); MAX. AND AVG. ITEMSETS SIZES (i_{max}, i_{avg}).

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

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 (the strategical impact of a building is less important after 10 minutes). Table V summarizes all characteristics of the datasets.

C. Run-time analysis and memory usage

We implemented our algorithms over the original C++ version of PREFIXSPAN [11], and experimented on a 1.8 GHz Intel Core i5 with 8 GB machine. Note that we released both the source-code and the datasets used in the following experiments [15][2]. We discuss running times of the proposed algorithms. Firstly, we consider the non-mirror databases given by transformed interaction databases $\mathcal{T}$ and signed databases $\mathcal{S}$ (since signed databases correspond to the general case whereas transformed databases are specific to non-mirror databases). For different minimum frequency thresholds σ, we present the runtime of PREFIXSPAN on $\mathcal{S}$ as a rough baseline (since it does not compute the balance of a pattern), and the runtimes of the three other algorithms on their respective data representation. It follows that our general algorithm BALANCESPAN is the only one able to be executed with lowest σ (Figure 1 (d), (e) and (f)). We report the same results for the general case with mirror datasets (i.e., $\mathcal{S}$ only for PvP, TvT and ZvZ) in Figure 1 (a), (b) and (c). BALANCESPAN clearly outperforms PREFIXSPANNAIVE, its only competitor (remembering that PREFIXSPAN is given as baseline since it does not compute the balances, and EMERGSPAN does not apply for mirror databases). Indeed, even if sometimes PREFIXSPANNAIVE seems to have the same run time as BALANCESPAN with high value of σ, it cannot reach lowest frequency thresholds σ. Note that on the figures, missing points correspond to unterminated runs when available memory is insufficient. Figure 3 shows

[2]Datasets, source-code and scripts are available on https://github.com/guillaume-bosc/BalanceSpan

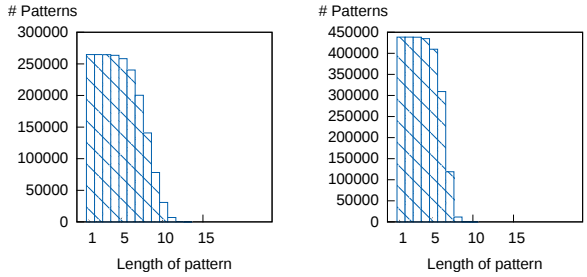


Fig. 3. Cumulative distribution of the length of the patterns (number of itemsets) for the PvT (left) and the ZvZ (right) datasets.

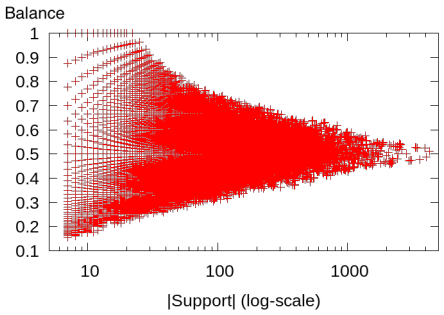


Fig. 4. Patterns support and balance for the dataset ZvZ

the distribution of the length of the extracted patterns for the PvT and ZvZ datasets.

The quantity of memory used is a very important aspect of our algorithms. Indeed, since the number of outputted patterns grows exponentially, see Figure 1, the memory usage becomes more and more important. Thus, the quantity of memory needed by our algorithms should be the as low as possible. In fact, the percentage of used nodes which are created in the tree structure are sensibly different for the algorithms. 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 $\mathcal{F}_s$). BALANCESPAN is the only algorithm that creates a node, and only one, for each pattern to be outputted, see Figure 2: in the best cases PREFIXSPANNAIVE have only the half of useful nodes (by definition). This number drops to 10% of useful nodes for very low supports on some datasets. For EMERGSPAN, it is worst, as only the direct predecessors of the leaves of the prefix tree are balanced patterns by definition.

D. Exploration of the extracted patterns

It is interesting to visualize the distribution of both the support and the balance of the patterns. Figure 4 gives such distribution for dataset ZvZ that enables 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 the equation $y = 0.5$ (where y is the vertical axis) gives almost a symmetry axis. Indeed, both a pattern and its dual do not necessarily have the same support. One can notice that empirically, there are high chances for a pattern with high frequency to have a fair balance around 0.5. This behavior applies for the other dataset and is what we could expect given the definition of the balance measure.

It is possible to query the set of extracted patterns in various ways. Indeed, the pattern mining task is related to the Knowledge Discovery in Databases (KDD) process that aims at extracting knowledge from data [8]. Data mining approaches and more precisely pattern mining approaches are a step of the KDD process that results in patterns from transformed data [9]. Our work is a pattern mining approach to study strategy balance that is applied to RTS games. Thus, one gets a set of patterns that are a generalization of the local strategies within the data. BALANCESPAN still requires an analysis once the patterns are outputted.

Exploring a large collection of patterns can be done in many ways. First, as illustrated hereafter, the expert can filter the collection with specific constraints such as a minimum number of itemsets, or specific items using regular expressions, etc. Second, the expert can introduce preferences as measures on the patterns (size, length, support, balance, etc.) that he wishes to minimize or maximize given his goals (the so-called *skypatterns* [16]). Indeed, one expert may favor highly balanced patterns with high support (probably the standard strategies), while another could be interested in maximizing the support while favoring patterns whose balance is closest to 0.5 (giving hints to possible game design problems). Finally, the discovery can be done interactively, through an interactive algorithm (not only the full KDD process [17]). The basic assumption is that the expert does not really know what he is looking for in the data, and guides the pattern discovery at each step of the algorithm (such as sequence expansion in our case). In all cases, the pattern language must be clearly defined and efficient algorithms proposed to compute the measure of interest, our main contribution.

We now provide a basic example of exploration by querying the pattern set. Out of the $43,610$ patterns for ZvZ with $\sigma = 0.001$, we can keep the patterns involving two players (containing both $+$ and $-$), which returns $40,674$ patterns. Then we restricted the set of patterns 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 players go to air units and the second to ground units, two known openings, with balance 0.47 and support 68.

The pattern mining task can be adapted in various ways,

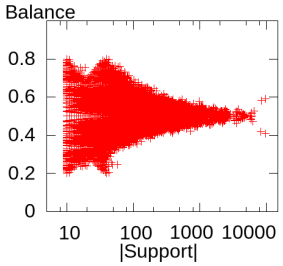


Fig. 5. Game openings support and balance for the dataset ZvZ

depending on which and how the basic actions are encoded in the sequence. Let us now sketch different scenarios.

1) Discovering strategy openings: Openings are the most well-known strategies and executed during the first five to ten minutes of a StarCraft II game. It is expected that openings are balanced to make the game enjoyable for the casual player, competitive for the professional, but also interesting to watch for the spectators [2]. We build our sequence databases with a set of items composed of tuples $(building, sign, i^{th} window)$, with fixed windows of 30 seconds by default, i.e., the i^{th} window contains the items performed between the $((i-1) \times 30)^{th}$ second and the $((i \times 30) - 1)^{th}$ second. We expect that openings are found as the more frequent patterns and being also balanced: Figure 5 shows the complete set of patterns for the ZvZ dataset which differs with Figure 4 by its skewness. We explored the resulting patterns with a game expert. When considering another dataset (PvZ), we obtain only 591 patterns with $\sigma = 0.05$. Top frequent patterns represent all well-known strategies: $s = \langle\{(Nexus, +, 5)\}\ \{(Gateway, +, 6)\ (Photon\ Cannon, +\ 6)\}\rangle$ represents a popular Protoss strategy, no matter the strategy of the opponent is. It is balanced ($balance(s) = 0.52$).

2) Discovering possible balance issues (hypotheses elicitation): The rules of the game are set by the editors and developers. However, such rules are not always fair and balanced, and such weaknesses can only be discovered after weeks. We asked an expert to highlight a well known imbalanced strategy. The so called *bunker rush* was used a lot by Terran players against their Zerg opponents. It consists of building in the early stage of the game a *bunker* near the opponent's base to put his economy and development in difficulty. After several complaints from the StarCraft II community, the rules changed on 10^{th} May 2012: a Zerg counter unit (*the queen*) has been slightly improved. Since then, this strategy stopped to be used for some time. Our approach should be able to reflect/discover that fact: we proceed as follows. We split dataset TvZ into two parts: the first one, called $\mathcal{S}_{before}$, contains replays that happened strictly before 10^{th} May 2012 ($17,171$ replays), and the second one, called $\mathcal{S}_{after}$, contains the replays that happened strictly after this change ($6,698$ replays). The mining of the dataset $\mathcal{S}_{before}$ (respectively $\mathcal{S}_{after}$) with a low $\sigma = 0.05$ returns $8,138$ patterns (resp. $7,735$). According to the experts, the *bunker* should be built during the sixth window of time (between 2'30" and 3' of the game). There are 20 (resp. 12) patterns that involve the item $(Bunker, c, 6)$ with $c \in \{+, -\}$. With $\mathcal{S}_{before}$ (resp. $\mathcal{S}_{after}$), the average value of the balance is 0.58 (resp. 0.51) with a standard deviation equals to 0.5 (resp. 1.6). This is clear that since the patch was released, this strategy has become balanced. Moreover, we can remark that this strategy is no longer used by players: in fact the number of extracted patterns related to this strategy decreases by 40% whereas the number of extracted patterns only decreases by 5% from $\mathcal{S}_{before}$ to $\mathcal{S}_{after}$. Thus, BALANCESPAN enables to see the impact of the release of patch, by analyzing the period before and after this key date.

3) On the diversity in mirror match-ups: It is more delicate to speak about balance when both players belong to the same faction (mirror match-ups): both players have access to the same strategies (same building, hence a symmetrical game). Let us observe for example the dataset PvP with a new vocabulary: items are tuples $(building, sign, i^{th} window, j^{building})$ where the last element records how many of this building were already made at the moment of the action (cumulative). Setting a minimal support $\sigma = 0.05$, we obtain 3418 patterns. We can find here the so-called *4 Gates* strategy through the pattern $s = \langle\{(Gateway, +, 3, 1)\ (Assimilator, +, 3, 1)\}\ \{(Cyb.Core, +, 4, 1)\}\ \{(Gateway, +, 7, 2)\ (Gateway, +, 7, 3)\ (Gateway, +, 7, 4)\}\rangle$ with $balance(s) = 0.59$. Such a high value may be surprising, but it reflects the effectiveness of this strategy, and consequently the poor diversity of the strategies used in the PvP match-up. It is an easy and non-risky strategy to apply: according to the expert, a player has better chances to win with this strategy against riskier strategies. After several recurrent complaints, nothing changed in the game until a new major update of the games (with new units and buildings). Since then strategies used in the PvP match up are more diversified and the *4 gates* strategy is rarely used.

VIII. RELATED WORK

Discovering patterns that highly distinguish a dataset from others (e.g., *win* labeled objects versus *lose* labeled objects) is an important task in machine learning and data mining [13]. One of the main reason is that such patterns enable the building of comprehensible classifiers [18]. 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.) [19]. The goal is to find good description 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 [13] like *version spaces* [20], *contrast sets* [21] and *subgroups discovery* [22] in machine learning, *emerging patterns* [12] in data-mining; or *no counter-example hypothesis* in formal concept analysis [19]. Our contribution in this field is to consider and compute efficiently a balance measure that existing methods can partially or non efficiently compute.

StarCraft II and other real time strategy games (RTS) in general, face several research challenges in artificial intelligence as deeply discussed in a recent survey [6]. Our work is related to the challenge that the authors of the survey qualify as *prior learning*, that is, techniques that can *"exploit available data such as existing replays [...] for learning appropriate strategies beforehand"*. Strategies in RTS game are complex and divided in several tasks, each bringing difficult problems. Several case-based reasoning approaches have been proposed, mainly to retrieve and adapt strategies (especially build orders) to be used then by an automated agent [23], [24]. Other kinds of approaches are also used for several prediction tasks. Predicting the opponent's production was studied with answer set programming [25], while learning transition probabilities within build orders was achieved with hidden Markov models [26]. Weber et al. described any past game by a vector of

building and upgrade timings: such features allow an accurate strategy prediction [27]. This comes with a limit: game logs are *a priori* labeled with a strategy using rules based on "manual" expert analysis. The same applies for opening prediction [28].

To discover strategies in large volumes of replays, avoiding to manually label game logs, knowledge discovery in database (KDD) methods are required, and especially pattern mining techniques. This was highlighted in the open problems category *Domain knowledge* of the recent survey mentioned before [6]: *"Is it possible to devise techniques that can automatically mine existing collections [...] and incorporate it into the bot?"*. We did not study in this article the second step (incorporation), but presented a way to extract efficiently such patterns and focused on*balance* issues for helping game designers. A long road remains to be able to select the right patterns to be used by artificial agents as discussed recently in [6], [29].

IX. Conclusion

We tackled the problem of mining frequent sequential patterns in real time strategy games whose balance measures provide meaningful insights on the strategies played and their ability of being in equilibrium or not. For that matter, we revisited the well known notions of discriminant pattern mining to provide efficient algorithms for the elicitation of balance hypotheses from the data.

From that, we presented several algorithms that enable (partially or not) dealing with interaction databases, and we showed that only BalanceSpan enables to deal with all datasets efficiently.

We empirically validated that the balance measure is able to distinguish balanced and imbalanced strategies. We believe that our approach can become a basic tool for *balance designers* when analyzing a subset of historical data of a game in beta phase, or even after its release, through an exploratory process (KDD and interactive mining). A major difficulty remains to select and construct features of interest from the game logs.

Acknowledgments

This work was partially supported by the MI CNRS Mastodons program that allowed the two last authors to visit the MIT Game Lab in July 2014.

References

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

[2] G. Cheung and J. Huang, "Starcraft from the stands: understanding the game spectator," in *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011*, 2011, pp. 763–772.

[3] M. Kaytoue, A. Silva, L. Cerf, W. M. Jr., and C. Raïssi, "Watch me playing, i am a professional: a first study on video game live streaming," in *Proceedings of the 21st World Wide Web Conference, WWW 2012*, 2012, pp. 1181–1188.

[4] A. Von Eschen, "Machine learning and data mining in call of duty," in *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, ser. LNCS 8724. Springer, 2014, an Industrial Invited Talk.

[5] M. A. Ahmad, B. Keegan, J. Srivastava, D. Williams, and N. S. Contractor, "Mining for gold farmers: Automatic detection of deviant players in mmogs," in *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, CSE 2009*, 2009, pp. 340–345.

[6] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in starcraft," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 5, no. 4, pp. 293–311, 2013.

[7] O. Missura and T. Gärtner, "Predicting dynamic difficulty," in *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011*, 2011, pp. 2007–2015.

[8] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The KDD process for extracting useful knowledge from volumes of data," *Commun. ACM*, vol. 39, no. 11, pp. 27–34, 1996.

[9] C. C. Aggarwal, *Data Mining - The Textbook*. Springer, 2015.

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

[11] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, "Prefixspan: Mining sequential patterns by prefix-projected growth," in *Proceedings of the 17th International Conference on Data Engineering*, 2001, pp. 215–224.

[12] G. Dong and J. Li, "Efficient mining of emerging patterns: Discovering trends and differences," in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999, pp. 43–52.

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

[14] M. Plantevit and B. Crémilleux, "Condensed representation of sequential patterns according to frequency-based measures," in *Advances in Intelligent Data Analysis VIII, 8th International Symposium on Intelligent Data Analysis, IDA 2009*, 2009, pp. 155–166.

[15] G. Bosc, M. Kaytoue, C. Raïssi, J. Boulicaut, and P. Tan, "Balancespan," 2015. [Online]. Available: https://github.com/guillaume-bosc/BalanceSpan

[16] A. Soulet, C. Raïssi, M. Plantevit, and B. Crémilleux, "Mining dominant patterns in the sky," in *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, 2011, pp. 655–664.

[17] M. van Leeuwen, "Interactive data exploration using pattern mining," in *Interactive Knowledge Discovery and Data Mining in Biomedical Informatics - State-of-the-Art and Future Challenges*, 2014, pp. 169–182.

[18] M. García-Borroto, J. F. M. Trinidad, and J. A. Carrasco-Ochoa, "A survey of emerging patterns for supervised classification," *Artif. Intell. Rev.*, vol. 42, no. 4, pp. 705–721, 2014.

[19] S. O. Kuznetsov, "Galois connections in data analysis: Contributions from the soviet era and modern russian research," in *Formal Concept Analysis, Foundations and Applications*, 2005, pp. 196–225.

[20] T. M. Mitchell, *Machine learning*, ser. McGraw Hill series in computer science. McGraw-Hill, 1997.

[21] S. D. Bay and M. J. Pazzani, "Detecting group differences: Mining contrast sets," *Data Min. Knowl. Discov.*, vol. 5, no. 3, pp. 213–246, 2001.

[22] F. Herrera, C. J. Carmona, P. González, and M. J. del Jesús, "An overview on subgroup discovery: foundations and applications," *Knowl. Inf. Syst.*, vol. 29, no. 3, pp. 495–525, 2011.

[23] D. W. Aha, M. Molineaux, and M. J. V. Ponsen, "Learning to win: Case-based plan selection in a real-time strategy game," in *Case-Based Reasoning, Research and Development, 6th International Conference, on Case-Based Reasoning, ICCBR 2005*, 2005, pp. 5–20.

[24] B. G. Weber and M. Mateas, "Case-based reasoning for build order in real-time strategy games," in *Proceedings of the Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2009*. The AAAI Press, 2009, pp. 106–111.

[25] M. Stanescu and M. Certicky, "Predicting opponent's production in real-time strategy games with answer set programming," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[26] E. W. Dereszynski, J. Hostetler, A. Fern, T. G. Dietterich, T. Hoang, and M. Udarbe, "Learning probabilistic behavior models in real-time strategy games," in *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2011*. The AAAI Press, 2011, pp. 20–25.

[27] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, CIG 2009*, 2009, pp. 140–147.

[28] G. Synnaeve and P. Bessière, "A bayesian model for opening prediction in RTS games with application to starcraft," in *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011*, 2011, pp. 281–288.
[29] M. Leece and A. Jhala, "Sequential pattern mining in starcraft: Brood war for short and long-term goals," in *Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2014*. The AAAI Press, 2014, pp. 281–288.