A Pattern Mining Approach to Study Strategy Balance in RTS Games
Guillaume Bosc, Philip Tan, Jean-François Boulicaut, Chedy Raïssi, Mehdi Kaytoue

To cite this version:


HAL Id: hal-01252728
https://hal.archives-ouvertes.fr/hal-01252728
Submitted on 8 Jan 2016

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

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A Pattern Mining Approach to Study Strategy Balance in RTS Games

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

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 analytics

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 over-viewing related work and concluding.
II. Prelimaries

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 $s = (X_1, \ldots, X_l)$ 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 $s = (X_1, \ldots, X_l)$ is a subsequence of a sequence $s' = (X'_1, \ldots, X'_{l'})$, denoted $s \subseteq s'$, if there exists $1 \leq j_1 < j_2 < \ldots < j_{l'} \leq l$ such that $X_1 \subseteq X'_{j_1}, X_2 \subseteq X'_{j_2}, \ldots, X_l \subseteq X'_{j_{l'}}$.

Definition (Support and frequency). Let $\mathcal{D}$ be a sequence database given in Table I, and $s \subseteq s'$ be the sequence database given in Table II. Given a sequence database $\mathcal{D}$, $\sup(s, \mathcal{D}) = |\{s' \in \mathcal{D} \mid s \subseteq s'\}|$ is the support of $s$. Its frequency is $\freq(s, \mathcal{D}) = \frac{|\sup(s, \mathcal{D})|}{|\mathcal{D}|}$.

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

In some cases, each sequence of $\mathcal{D}$ is associated to a class label. Let $\class : \mathcal{D} \rightarrow \{+,-\}$ 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 $\mathcal{D}^+$ to $\mathcal{D}^-$ ($x \neq y$ and $x,y \in \{+,-\}$), is given by

$$\text{growth-rate}(s, \mathcal{D}^+, \mathcal{D}^-) = \frac{|\sup(s, \mathcal{D}^+)|}{|\mathcal{D}^+|} \times \frac{|\mathcal{D}^-|}{|\sup(s, \mathcal{D}^-)|}$$

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 $s = (abc)$, we have $s \subseteq s_1, s \subseteq s_4, s \nsubseteq s_2, s \nsubseteq s_3$. With $\sigma = \frac{3}{4}$, $(acc)$ is frequent, $(a\{bc\}a)$ is not. We have $\text{growth-rate}((eb), \mathcal{D}^+, \mathcal{D}^-) = \frac{3}{2} \times \frac{3}{2} = 2$, i.e., $(eb)$ is twice more present in class $-$ than in class $+$.

### Table I

<table>
<thead>
<tr>
<th>id</th>
<th>$s \in \mathcal{D}$</th>
<th>class($s$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$(a{bc}{ac}d{cf})$</td>
<td>+</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$({ad}c{be}{a})$</td>
<td>+</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$({ec}{eb}{df}eb)$</td>
<td>-</td>
</tr>
<tr>
<td>$s_4$</td>
<td>$(eq{af}{bc})$</td>
<td>-</td>
</tr>
</tbody>
</table>

### Table II

<table>
<thead>
<tr>
<th>id</th>
<th>Interaction sequence</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$(p_1, a){(p_1, b){p_1, c}{p_2, c}}{(p_2, a){p_1, d}}$</td>
<td>$p_1$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$(p_3, a){(p_3, b){p_3, c}{p_3, d}}{(p_1, b){p_1, c}}$</td>
<td>$p_3$</td>
</tr>
</tbody>
</table>
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 \( sup(s, \mathcal{T}^{I_1}) \) and \( sup(s, \mathcal{T}^{I_2}) \) 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. \( \sigma \) 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 \( s \in \mathcal{F}_t \), \( freq(s, \mathcal{T}^{I_1}) \geq \sigma \) and \( freq(s, \mathcal{T}^{I_2}) \geq \sigma \) (implying \( freq(s, \mathcal{T}) \geq \sigma \)) and the balance measure is computed and given by:

\[
balance(s, \mathcal{T}^k) = \frac{|sup(s, \mathcal{T}^k)|}{|sup(s, \mathcal{T}^I)| + |sup(s, \mathcal{T}^T)|}
\]

**Remark.** The balance measure is a normalized version of the growth rate measure 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 \( \sigma = 0.2 \), \( s = \langle ab \rangle \{c\} \) is a frequent balanced pattern since \( freq(s, \mathcal{T}^{I_1}) = \frac{2}{3} \) and \( freq(s, \mathcal{T}^{I_2}) = \frac{1}{3} \). Moreover, balance(\( s, \mathcal{T}^{I_1} \)) = \( \frac{2}{3} \) and balance(\( s, \mathcal{T}^{I_2} \)) = \( \frac{1}{3} \). It means that \( s \) wins two times more for the type 1 of player than for the type 2.

<table>
<thead>
<tr>
<th>id</th>
<th>Interaction sequence</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>( \langle ab \rangle {c} )</td>
<td>( \mathcal{I}_1 )</td>
</tr>
<tr>
<td>s2</td>
<td>( \langle ab \rangle {a} {c} )</td>
<td>( \mathcal{I}_1 )</td>
</tr>
<tr>
<td>s3</td>
<td>( \langle abc \rangle {c} )</td>
<td>( \mathcal{I}_2 )</td>
</tr>
<tr>
<td>s4</td>
<td>( \langle ac \rangle {a} {c} )</td>
<td>( \mathcal{I}_1 )</td>
</tr>
<tr>
<td>s5</td>
<td>( \langle c \rangle {b} {ac} )</td>
<td>( \mathcal{I}_2 )</td>
</tr>
</tbody>
</table>

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 \( \mathcal{I}_s = \text{Actions} \times \{-, +\} \) 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 \( \mathcal{I}_s = \text{Actions} \times \{-, +\} \) 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

\[
dual(a, c) = \dual(a^c) = \langle a, \{+, -\} \backslash c \rangle = a^{(+,-)} \backslash c
\]

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 \( s = \langle X_1, X_2, \ldots \rangle \) a sequence over \( \mathcal{I}_s \):

\[
dual(X) = \{\dual(x) | \forall x \in X\}
\]

\[
dual(s) = \{\dual(X_1), \dual(X_2), \ldots\}
\]

**Example.** In Table IV, we have \( \mathcal{I}_s = \{a, b, c, d\} \times \{-, +\} \) and \( \dual(a^-) = a^+ \) 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 \( S \). The balance measure of \( s \) is

\[
balance(s) = \frac{|\text{sup}(s, S)|}{|\text{sup}(s, S)| + |\text{sup}(\text{dual}(s), S)|}
\]

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 \( s \in S \). Consider the following example: \( S = \{\{a^+b^-\}, \{a^-b^+\}\}, \{\{a^-b^-\}\} \) with \( \sigma = \frac{3}{4} \). The sequence \( s = \{a^-b^-\} \) is a frequent sequential pattern, and \( |\text{sup}(s, S)| = 1 \). We have also \( |\text{sup}(\text{dual}(s), S)| = |\text{sup}(\text{dual}(\{a^-b^+\}), S)| = 2 \). Hence, \( balance(s) = \frac{1}{1+2} = \frac{1}{3} \). However, since \( s \) and \( \text{dual}(s) \) 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

\[
balance_gen(s) = \frac{|\text{sup}(s, S)| - |\text{sup}(|\text{dual}(s), S)|}{|\text{sup}(s, S)| + |\text{sup}(|\text{dual}(s), S)|}
\]

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

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

Example. Table IV gives a signed interaction database \( S \) obtained from an arbitrary \( \mathcal{R} \). With \( \sigma = \frac{1}{4} \), \( s = \{a^+c^-\} \) appears two times, its dual appears only once, hence \( balance(s) = \frac{1}{3} \).

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 \( \text{common}(s) = \emptyset \) 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 \textsc{Pattern-Growth} and its associated algorithm \textsc{PrefixSpan} [11].

A. The \textsc{PrefixSpan} algorithm

Given a sequence database \( D \) over items \( I \) and a minimum frequency threshold \( \sigma \), \textsc{PrefixSpan} outputs all frequent sequential patterns and only them [11]. Firstly, the database \( D \) is scanned once to find all the frequent items from \( I \), 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 \( a \in I \)), one retains only sequences of \( D \) 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 \( D_{\langle a \rangle} \). 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 \( [\sigma \times |D|] \) 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 \( s \circ a \), and appending \( a \) within the last itemset of \( s \), written \( s \circ a \circ a \) (\( a \) denotes an extension in general). At the end, the pattern tree structure is explored and each node outputs a pattern.

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

B. Mining Balanced Patterns with \textsc{EmergSpan}

Given a sequential database over an arbitrary \( I \), 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 step [14]. Based on \textsc{PrefixSpan}, \textsc{EmergSpan} works as follows. For each sequence of the database \( T \), 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, 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 $\{\{ab\}ac\mathcal{I}_1\}$. 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 $\langle X_1 \ldots X_t \rangle$ only requires to backtrack to the parent (i.e., $\langle X_1 \ldots X_{t+1} \rangle$) to get its support to evaluate the balance. So, when EMERGSPAN deals with the sequential pattern $s = \{\{ab\}c\mathcal{I}_1\}$, 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 $s' = \{\{ab\}c\}$: $\text{balance}(s) = \frac{\text{sup}(s, T)}{\text{sup}(s', T)} = 0.67$.

C. Mining Balanced Patterns with PREFIXSPANNAIVE

Now we consider a signed interaction database $\mathcal{S}$ over $\mathcal{I}_s$ and a minimal frequency threshold $\sigma$. 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 $\text{dual}(q)$ by searching among the children of $\text{dual}(p)$. If $\text{dual}(q)$ exists, the pattern $q$ is outputted, $\text{dual}(q)$ is flagged as already outputted (redundant pattern) and the process recursively continues. Otherwise, the algorithm backtracks. In this way, $q$ and $\text{dual}(q)$ 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 $\text{common}(q)$ which is stored for each node. The proof of the completeness and correctness of PREFIXSPANNAIVE for extracting all balanced patterns without redundant is direct: first, PREFIXSPAN extracts all frequent patterns, thus any pattern $s$ and its dual $\text{dual}(s)$ 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 $\text{dual}(s)$ (if frequent), we ensure no redundant patterns.

Example. PREFIXSPANNAIVE requires a signed interaction 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 $s = \langle \rangle$ (\text{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 $(a^-)$ is the dual of $(a^+)$, and calls $\text{PREFIXSPANNAIVE}(a^+, \mathcal{S}(a^+), \sigma)$. Then, PREFIXSPANNAIVE searches for frequent items in $\mathcal{S}(a^+)$: e.g., $c^-$ is frequent. So, it will search in the children of $(a^-)$ if there is $c^+$, but at this step, the node of $(a^-)$ has not been explored yet. Thus, when $(a^-)$ is explored, PREFIXSPANNAIVE extracts frequent items in $\mathcal{S}(a^-)$: e.g., $c^+$ is frequent. So it will search among the children of $\text{dual}(a^-)$ if $\text{dual}(c^+)$ exists: $(a^-c^+)$ 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 then, i.e. correct and complete), we

---

**Algorithm: PREFIXSPAN**

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

**Algorithm: EMERGSPAN**

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

---

---
propose the \textsc{BalanceSpan} approach. The general idea is the following: instead of considering each item \( a \in \mathcal{I} \) as an extension on sequence \( s \) leading to a new projected database \( S_{|s \cdot a} \) and consequently a new node in the pattern tree, we consider simultaneously an item and its dual, hence two projected databases \( S_{|s \cdot a} \) and \( S_{\text{dual}(s) \cdot \text{dual}(a)} \) are related to a single node (this is done for both kinds of extensions \( o_1 \) and \( o_2 \)). Thus, it ensures that no redundant patterns are generated, since both a sequence \( s \) and \( \text{dual}(s) \) are generated at the same node, and it allows to compute \( \text{balance}(s) \) (or \( \text{balance}(\text{dual}(s)) \)) directly if and only if both \( s \) and \( \text{dual}(s) \) are frequent. It follows that \textsc{BalanceSpan} produces a correct and complete collection of frequent balanced patterns.

\textbf{Example.} \textsc{BalanceSpan} also requires a signed interaction database, but contrary to \textsc{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 \textsc{PrefixSpanNaive}, \textsc{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 \((a^+, a^-)\). The next step calls \textsc{BalanceSpan} on this new node. So it searches for frequent items in the projected database \( S_{\langle a^+ \rangle} \): e.g., \( c^- \). Thus it checks if \( \text{dual}(c^-) \) is frequent in \( S_{\langle a^- \rangle} \): the node containing the couple \( \langle a^+ c^- \rangle, \langle a^- c^+ \rangle \) is created and explored in the next step.

\section{VII. Experiments}

\subsection{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 replay.

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.

\subsection{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\footnote{http://wiki.teamliquid.net/starcraft2/Replay\_Websites}. 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

\begin{algorithm}[H]
\caption{\textsc{PrefixSpanNaive}}
\begin{algorithmic}[1]
\Require A sequence \( s \), the \( s \)-projected signed DB \( S_{|s \cdot a} \), and a minimum threshold \( \sigma \)
\Ensure The frequent sequential pattern set \( F \)
\begin{algorithmic}
\State 1: Compute \( \text{balance}(s) \) thanks to \( \text{dual}(s) \)
\State 2: \( F \leftarrow \{s\} \)
\State 3: scan \( S_{|s \cdot a} \) once, find every frequent item \( a \) such that (a) \( s \) can be extended to \( (s \cdot o_1) \), or (b) \( s \) can be extended to \( (s \cdot o_2) \)
\State 4: if no valid \( a \) available then return \( F \)
\State 6: end if
\State 7: for all valid \( a \) do
\State 8: (a) Search for \( \text{dual}(s \cdot o_1) \) among children of \( \text{dual}(s) \)
\State 9: if \( \text{dual}(s \cdot o_1) \) exists then
\State 10: Link \( \text{dual}(s \cdot o_1) \) to \( s \cdot o_1 \)
\State 11: \( F \leftarrow F \cup \text{PrefixSpanNaive}(s \cdot o_1, S_{|s \cdot o_1 \cdot a}, \sigma) \)
\State 12: end if
\State 13: (b) Search for \( \text{dual}(s \cdot o_2) \) among children of \( \text{dual}(s) \)
\State 14: if \( \text{dual}(s \cdot o_2) \) exists then
\State 15: Link \( \text{dual}(s \cdot o_2) \) to \( s \cdot o_2 \)
\State 16: \( F \leftarrow F \cup \text{PrefixSpanNaive}(s \cdot o_2, S_{|s \cdot o_2 \cdot a}, \sigma) \)
\State 17: end if
\State 18: end for
\State 19: return \( F \)
\end{algorithmic}
\end{algorithm}

\begin{algorithm}[H]
\caption{\textsc{BalanceSpan}}
\begin{algorithmic}[1]
\Require A sequence \( s \), its dual sequence \( s' = \text{dual}(s) \), the \( s' \)-projected signed DB \( S_{|s' \cdot a'} \), and a minimum threshold \( \sigma \)
\Ensure The frequent balanced pattern set \( F \)
\begin{algorithmic}
\State 1: Compute \( \text{balance}(s) \)
\State 2: \( F \leftarrow \{(s, s')\} \)
\State 3: scan \( S_{|s' \cdot a'} \) once, find every frequent item \( a \) such that (a) \( s' \) can be extended to \( (s' \cdot o_1) \), or (b) \( s' \) can be extended to \( (s' \cdot o_2) \)
\State 4: scan \( S_{|s' \cdot a'} \) once, find every frequent item \( b \) such that (a) \( s' \) can be extended to \( (s' \cdot o_1 b) \), or (b) \( s' \) can be extended to \( (s' \cdot o_2 b) \)
\State 5: if no valid \( a \) or \( b \) available then return \( F \)
\State 7: end if
\State 8: for all valid \( a \) and \( b \) such that \( a = \text{dual}(b) \) do
\State 9: (a) \( F \leftarrow F \cup \text{BalanceSpan}(s \cdot o_1, s' \cdot o_1 b, S_{|s \cdot o_1 \cdot a}, S_{|s' \cdot o_1 \cdot b}, \sigma) \)
\State 10: (b) \( F \leftarrow F \cup \text{BalanceSpan}(s \cdot o_2, s' \cdot o_2 b, S_{|s \cdot o_2 \cdot a}, S_{|s' \cdot o_2 \cdot b}, \sigma) \)
\State 11: end for
\State 12: return \( F \)
\end{algorithmic}
\end{algorithm}
guillaume-bosc/BalanceSpan

Firstly, we consider the non-mirror databases given experiments \[15\]. 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 \(\sigma\), we present the runtime of \textsc{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 \textsc{BalanceSpan} is the only one able to be executed with lowest \(\sigma\) (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). \textsc{BalanceSpan} clearly outperforms \textsc{PrefixSpanNaive}, its only competitor (remembering that \textsc{PrefixSpan} is given as baseline since it does not compute the balances, and \textsc{EmergSpan} does not apply for mirror databases). Indeed, even if sometimes \textsc{PrefixSpanNaive} seems to have the same run time as \textsc{BalanceSpan} with high value of \(\sigma\), it cannot reach lowest frequency thresholds \(\sigma\). Note that on the figures, missing points correspond to unterminated runs when available memory is insufficient. Figure 3 shows

![Fig. 1. Run time and number of patterns.](image)

![Fig. 2. Percentage of used nodes in the PvP dataset.](image)

![Table V. Datasets: sequence and item counts; max. and avg. sequence sizes (\(s_{max}, s_{avg}\)); max. and avg. itemsizes sizes (\(i_{max}, i_{avg}\)).](image)
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 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 $\mathcal{F}_L$ (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 $\sigma$ (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 
otice 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. $\langle$SpawnPool$^+$, SpawnPool$^-$, SpCrawler$^+$, 
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,
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 \((\text{building}, \text{sign}, i^{th} \text{window})\), 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: \(s \in \{\{(\text{Nexus}, +, 5)\}, \{(\text{Gateway}, +, 6)\}, \{\text{Photon Cannon}, +, 6\}\}\) represents a popular Protoss strategy, no matter the strategy of the opponent is. It is balanced \((\text{balance}(s) = 0.52)\).

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 \(S_{\text{before}}\), contains replays that happened strictly after this change \(10^{th}\) May 2012 (17, 171 replays), and the second one, called \(S_{\text{after}}\), contains the replays that happened strictly after this change (6, 698 replays). The mining of the dataset \(S_{\text{before}}\) (respectively \(S_{\text{after}}\)) 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 \((\text{Bunker}, c, 6)\) with \(c \in \{+, -\}\). With \(S_{\text{before}}\) (resp. \(S_{\text{after}}\)), 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 \(S_{\text{before}}\) to \(S_{\text{after}}\). 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 \((\text{building}, \text{sign}, i^{th} \text{window}, \text{standing})\) 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 \(s = \{(\text{Gateway}, +, 3, 1)\} \{(\text{Assimilator}, +, 3, 1)\} \{(\text{Cyb.Core}, +, 4, 1)\} \{(\text{Gateway}, +, 7, 2)\} \{(\text{Gateway}, +, 7, 3)\} \{(\text{Gateway}, +, 7, 4)\}\) 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
