World Wide Web
DOI 10.1007/s11280-015-0350-4

# An effective contrast sequential pattern mining approach to taxpayer behavior analysis

**Zhigang Zheng[1] · Wei Wei[1] · Chunming Liu[1] · Wei Cao[1] · Longbing Cao[1] · Maninder Bhatia[2]**

Received: 7 January 2014 / Revised: 29 January 2015 / Accepted: 20 April 2015


**Abstract** Data mining for client behavior analysis has become increasingly important in business, however further analysis on transactions and sequential behaviors would be of even greater value, especially in the financial service industry, such as banking and insurance, government and so on. In a real-world business application of taxation debt collection, in order to understand the internal relationship between taxpayers' sequential behaviors (payment, lodgment and actions) and compliance to their debt, we need to find the *contrast* sequential behavior patterns between compliant and non-compliant taxpayers. Contrast Patterns (CP) are defined as the itemsets showing the difference/discrimination between two classes/datasets (Dong and Li, 1999). However, the existing CP mining methods which can only mine itemset patterns, are not suitable for mining sequential patterns, such as time-ordered transactions in taxpayer sequential behaviors. Little work has been conducted on Contrast Sequential Pattern (CSP) mining so far. Therefore, to address this issue, we develop a CSP mining approach, $eCSP$, by using an effective CSP-tree structure, which improves the PrefixSpan tree (Pei et al., 2001) for mining contrast patterns. We propose some heuristics and interestingness filtering criteria, and integrate them into the CSP-tree seamlessly to reduce the search space and to find business-interesting patterns as well. The performance of the proposed approach is evaluated on three real-world datasets. In addition, we use a case study to show how to implement the approach to analyse taxpayer behaviour. The results show a very promising performance and convincing business value.

**Keywords** Contrast pattern · Sequential pattern · Client behavior analysis

✉ Zhigang Zheng
zhigang.zheng@uts.edu.au

[1] Advanced Analytics Institute, Faculty of Engineering and Information Technology, University of Technology, Sydney, Australia

[2] Australian Taxation Office, Sydney, Australia

Published online: 03 June 2015

## 1 Introduction

Behavior analysis focuses on human activities, such as transactions and interactions, under specific environments and backgrounds. It has been increasingly highlighted in many fields, such as customer relationship management [20], web search and usage [1, 3], the financial services industry, government [27] and so on. Client behavior plays a very important role as an internal driving force for many business issues. Hence, it is critically important to deeply understand client behavior in order to provide customized services to specific groups of clients [8]. Typically, traditional client behavior analysis is based on client static attributes, i.e., demographic data. However, sequential behaviour, which has not received much research attention to date, is much more valuable. For example, in real-world business, it is essential to understand the relationship between taxpayers' lodgment/payment (sequential) behaviors and their compliance to their debt, which involves finding the *contrast sequential behavior patterns* between compliant and non-compliant taxpayers.

Contrast Patterns (CP), which are the itemsets showing the discrimination between two classes/datasets, can solve a similar problem in non-sequence data, such as medical test comparisons, purchase behavior analysis, etc [10].

For example, in client churn analysis, there are two datasets: one dataset contains records of clients who have churned, and the other one contains clients who are still active. Suppose that each dataset has 25,000 records. After CP mining, we find a pattern <HasPhoneNumber?=No, NumOfTenureYears=3-5, Changed_Sales?=Yes> (this means a client has no phone number in the system, it has been 3-5 years since he/she had purchased the product, and the client has changed providers), which has 1,470 records/support from churn clients and no support from active clients. This means that those clients who are covered by this pattern all churned. From a business perspective, this is an very useful rule which can distinguish churn customers and active ones.

CP has been proven to be very explainable and efficient on a two-classes comparison and has a high rate of prediction accuracy [10, 12]. The concept was firstly proposed by Dong et al. [10] and was later studied by [5, 12, 13, 24], etc. However, (1) existing CP mining methods can solve itemset pattern mining problems but not sequential pattern mining. Since sequence data are a special form of data in real applications, such as in protein comparisons, text mining, web click analysis, etc. current CP mining methods are not suitable for mining sequential patterns. Sequence mining would meet more huge search space and much higher complexity compared to traditional relational data or itemset mining. (2) Existing sequential pattern mining methods, i.e., FreeSpan [15], PrefixSpan [22], SPADE [26], SPAM [4], etc. are not suitable for Contrast Sequential Pattern (CSP) mining, since they mostly reduce the search space by the Apriori anti-monotone property [2], which is not applicable to CSP [10]. To address these two problems, we propose a CSP mining approach.

It is challenging to mine CSP for the following reasons:

- Firstly, when the support threshold is low when using existing frequent patterns mining methods, the search space grows enormously.
- Secondly, the Apriori property is popular and is quite effective for pruning in frequent pattern mining. However, since it doesn't hold for CSP, it will be very challenging to find an efficient approach to reduce the search space in CSP mining.
- Thirdly, previous CP mining approaches often produce a large number of patterns which makes it difficult to choose interesting ones manually in real-world applications. Usually, post-processing would be applied to filter out uninteresting ones.

In order to address this need, we propose a new CSP mining algorithm, which is called *eCSP*. It can mine discriminating sequential patterns between two sequence datasets, which shows two different classes/types of population, to find the sequential patterns occurring frequently in one dataset, but infrequently in another one.

The main contributions of this paper are as follows.

(1) An effective CSP mining algorithm is built on a CSP-tree to search all possible sequence spaces to find CSP candidates, while keeping their corresponding class information in the projected databases; (2) Several pruning methods, based on some interestingness measures, are proposed and integrated in the algorithm to significantly reduce the search space, and output the most interesting patterns directly to target real-life business problems; (3) The proposed algorithm is examined using three real-world datasets, the results showing the high efficiency of the approach and its robust scalability performance; and (4) We implement a case study on tax debt collection optimization by using taxpayers sequence behavior data (i.e., lodgments, payments and actions transactions) to find which behavior patterns are likely to result in compliant debt cases, which do not require any action to be taken, in order to save resources and maximize the profit of the business.

This paper is organized as follows. Section 2 reviews the related work. Section 3 describes the problem of CSP mining and client behavior analytics. Section 4 details the eCSP approach. The experimental evaluation and a case study on taxpayer behavior analysis are presented in Section 5. Section 6 concludes the paper and suggests directions for future research.

## 2 Related work

CP mining has been widely studied in terms of Emerging Pattern (EP) [10], Jumping Emerging Patterns (JEP) [14] and the contrast capturing method [18]. EP [10] was proposed to find patterns whose supports increase significantly from one dataset to another. The significance of this increase is defined as the *Growth Rate*, which is the ratio between two support values on two datasets. Given a growth rate threshold, the target is to find all the patterns whose growth rates are larger than the threshold. Most current CP mining approaches use itemset mining, which are not able to solve the sequence mining problem.

CP is a very interesting and feasible methodology for discovering multi-attribute contrasts between data classes, especially when dealing with real-life applications. Many researchers [5, 12, 24] investigated this area after EP [10] was proposed. The work in [14] introduces JEPs whose supports increase abruptly from zero in one data set to non-zero in another. Dong et al. [12] and Ramamohanarao et al. [24] built classifiers using EPs or JEPs, and claimed that they outperformed most classifiers, such as CMAR [17] and C4.5 [23], in terms of accuracy. Dong et al. [12] partitioned the dataset to obtain the opposite class, or the target class, and mined them separately to obtain "sharp" EPs, and then adopted a scoring mechanism for classification.

Most methods for mining CPs are based on borders in terms of pattern output. Dong et al. [10, 11] employed the Max-Miner [6] algorithm whilst work in [21] is also applicable for generating large borders. Mannila et al. [19] restricts border use to subset closed collections. Bailey et al. [5] investigated interval closed collections which contained only minimal elements derived from the base collections.

Most CP mining approaches are not applicable for sequence data. Chan et al. [7] and Ji et al. [16] conducted valuable work on sequence CP mining, and Wang et al. [25] proposed

methods to distinguish sequential patterns which focus on density-aware patterns with gap constraints. Emerging Substring [7] refers to strings of items used to differentiate between two classes of sequences, and a suffix tree was used to store all the substrings. But the work of [16] can only deal with sequences of items rather than sequences of itemsets. In 2005, Ji et al. [16] introduced a new type of CP, called a Minimal Distinguishing Subsequence (MDS), which is applicable for sequence data. MDS is a minimal subsequence that occurs frequently in one class of sequences and infrequently in sequences of another class. Ji et al. [16] also tried to reduce the search space by adding minimum and maximum gap constraint, as well as a maximum length constraint. But more effective strategies, such as pruning approaches, are still required to reduce the search space. Compared to the work of [16, 25] and [7], our proposed approach can solve itemset sequence mining, with specific business-interesting filtering criteria integrated in the mining process, rather than simply post-mining for interesting contrast patterns.

## 3 Problem statements

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of ***items***. An ***itemset*** is a subset of $I$. A ***sequence*** $s$ is an ordered list of itemsets, $s =< e_1\ e_2\ ...\ e_n >$, where each $e_j$, $1 \leq j \leq n$, is an ***element***. An *element* $e_j$ ($1 \leq j \leq$k) consists of one or more items.

For example, sequence $< a\ c\ (cd)\ f >$ consists of four elements and *(cd)* is an element which includes two items.

The *length* of a sequence is the number of items in the sequence. A sequence with k items is called a *k-item sequence*. A sequence $s_r =< e_{r_1}\ e_{r_2}\ ...\ e_{r_m} >$ is a ***subsequence*** of another sequence $s_p =< e_{p_1}\ e_{p_2}\ ...\ e_{p_n} >$, if there exists $1 \leq i_1 \leq i_2 \leq ... \leq i_m \leq p_n$, $e_{r_1} \subseteq e_{p_{i_1}}, e_{r_2} \subseteq e_{p_{i_2}}, ..., e_{r_m} \subseteq e_{p_{i_m}}$. We can also say $s_p$ *contains* $s_r$.

For example, <a>, <c> and $< (cd)\ f >$ are some sub-sequences of $< a\ c\ (cd)\ f >$.

A ***sequence database*** $D$ is the source dataset for modeling. Any sequence in sequence database is a $data\ sequence$, denoted by $ds$. So in a sequence database $D = \{ds_i, 0 < i < n\}$, $n$ is the number of data sequences in a sequence database. The ***support*** of sequence $s$, denoted as $sup(s)$, is the number of data sequences which contain $s$, i.e., $sup(s) =| \{ds, ds \in D \wedge (s \subseteq ds)\} |$, $D$ is the sequence database. Or we use $sup(s, c_i) =| \{ds, ds \in D_{c_i} \wedge (s \subseteq ds)\} |$ if $D_{c_i}$ is a sequence database which only contains the data sequences with $class = c_i$. $min\_sup$ is a minimum support threshold which is predefined by users. $ds$ is a data sequence in the sequence database. Sequence $s$ is called a ***frequent*** *sequential pattern* if $sup(s) \geqslant min\_sup$. By contrast, $s$ is *infrequent* if $sup(s) < min\_sup$.

Given a sequence $\alpha =< e_1 e_2 ... e_n >$, a sequence $\beta =< e'_1 e'_2 ... e'_m >$ $(m \leqslant n)$ is called a ***prefix*** of $\alpha$ if and only if (1) $e'_i = e_i$ for $(i \leqslant m - 1)$; (2) $e'_m \subseteq e_m$. If there is more than one item in an element, we use _ to represent the prefix to distinguish this from the single item element [22].

For example, given $s =< a\ c\ (cd)\ f >$, its prefix could be $< a >$, $< a\ c >$, $< a\ c(c\_) >$, $< a\ c(cd) >$, etc. $< a\ c(c\_) >$ is one of the prefixes since $c$ and $d$ are in a same element, so we use $c\_$ to differentiate it from $c$.

Given a prefix $pref$ of a sequence $s$, $proj$ is its corresponding ***projected subsequence***, denoted by $proj(s, pref)$, if: (1) $proj$ is composed of all itemsets which occur after $pref$; (2) There exists no proper super-sequence $proj'$ of $proj$ such that $proj'$ is a subsequence of $s$ and also has prefix $pref$. Given a prefix $pref$, its *projected database* is composed of all projected subsequences of the data sequences which contain $pref$, and is denoted by $D_{proj} = \{proj(ds_i, pref), ds_i \in D, pref \subseteq ds_i\}$.

For example, given $s =< a\ c(cd)f >$ and $pref =< a\ c >$, the corresponding projected subsequence is $< (cd)f >$, or $proj(s, pref) =< (cd)f >$; if $pref' =< a\ c\ (c\_) >$, then $proj(s, prof') =< (\_d)\ f >$

### 3.1 Contrast sequential patterns

**Definition 1** ***Growth Rate***

Assume that we are given two sequence datasets $D_1$ and $D_2$, which belong to two different classes, $c_1$ and $c_2$, respectively. Given a sequential pattern $X$, its *growth rates* from $D_2$ to $D_1$ and from $D_1$ to $D_2$, denoted by $GR_{c_1}(X)$ and $GR_{c_2}(X)$, are defined as:

$$GR_{c_1}(X) = \begin{cases} \infty, & if\ \ sup(X, c_2) = 0\ \&\ sup(X, c_1) \neq 0 \\ \frac{sup(X,c_1)/|D_1|}{sup(X,c_2)/|D_2|}, & otherwise \end{cases} \tag{1}$$

$$GR_{c_2}(X) = \begin{cases} \infty, & if\ \ sup(X, c_1) = 0\ \&\ sup(X, c_2) \neq 0 \\ \frac{sup(X,c_2)/|D_2|}{sup(X,c_1)/|D_1|}, & otherwise \end{cases} \tag{2}$$

$$GR_{c_1}(X) * GR_{c_2}(X) = 1 \tag{3}$$

If $GR_{c_1}$ is greater than 1, it has a higher probability to be $c_1$ in the datasets, otherwise, it is more likely to be $c_2$.

*Example 1* dataset $D_1$ has 1000 records, $D_2$ has 100 records.

Given a pattern $A$, if $sup(A, c_1) = 100, sup(A, c_2) = 50$, then $GR_{c_1}(A) = (100/1000)/(50/100) = 0.2$ and $GR_{c_2}(A) = 5$.

Given a pattern $B$, if $sup(B, c_1) = 500$, $sup(B, c_2) = 5$, then $GR_{c_2}(B) = (5/100)/(500/1000) = 0.1$ and $GR_{c_1}(B) = 10$.

**Definition 2** ***Contrast Rate***

$ContrastRate$ (CR) represents the differences between two classes. For a sequential pattern $X$, its contrast rate is defined as follows:

$$CR(X) = \begin{cases} GR_{c_1}(X), & if\ GR_{c_1}(X) \geq 1 \\ GR_{c_2}(X), & if\ GR_{c_2}(X) \geq 1 \\ \infty, & if\ GR_{c_1}(X) = 0\ or\ GR_{c_2}(X) = 0 \end{cases} \tag{4}$$

For example, in the above example, $CR(A) = 5$ and $CR(B) = 10$. Pattern $B$ shows a higher distinguishing capability.

**Definition 3** ***Contrast Sequential Pattern (CSP)***

Given a threshold of contrast rate $min\_cr$ and a sequential pattern $X$, if $CR(X) \geqslant min\_cr$, then $X$ is a *Contrast Sequential Pattern*.

**Definition 4** ***Jumping Contrast Sequential Pattern (JCSP)***

Given a pattern $X$, if $CR(X) = \infty$, then we call it a *Jumping Contrast Sequential Pattern*.

CSP mining aims to find the CSPs with higher CR than a given CR threshold $min\_cr$. As shown in Figure 1, if we set a threshold $min\_cr = 3$, the patterns located in areas A and B are the target. Area $A$ represents all patterns with 3+ times $c_2$ support (%) than $c_1$ support (%). Area $B$ represents all patterns with 3+ times $c_1$ support (%) than $c_2$ support (%).

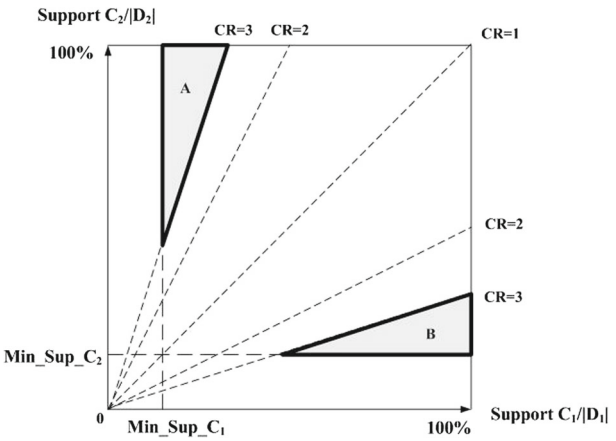



**Figure 1** Contrast rate of CSP

**Apriori property and CSP mining** The Apriori property can be simply described as follows: a sequence $s$ is not frequent if any of its sub-sequences $s'$ are not frequent, namely, $sup(s') \geqslant sup(s)$. However, CSP mining does not satisfy the property, as shown in the following Example 2.

*Example 2* We have two datasets $D_1$ and $D_2$, which represent class $c_1$ and $c_2$ respectively. $D_1$ has 1,000 records and $D_2$ also has 1,000 records. Given two patterns $s_1 =< a\ b >$ and $s_2 =< a\ b\ c >$, $s_1$ is a sub-sequence of $s_2$:

If $sup(s_1, c_1) = 80$, $sup(s_1, c_2) = 100$, $CR(s_1) = 1.25$; and $sup(s_2, c_1) = 10$, $sup(s_2, c_2) = 60$, $CR(s_2) = 6$. If we set $min\_cr = 5$, then $s_2$ is a CSP, and $s_1$ is not. In this case, $s_1$ is a subsequence of $s_2$, but $s_2$ is a CSP but $s_1$ is not, which doesn't follow Apriori property.

### 3.2 Client behavior analysis

As pointed out in the existing studies, a huge amount of client data, including clients' information and activities, is stored in business systems, which can be summarized as follows:

(1) *"Who"* represents a customer who owns a lot of attributes, such as age, gender, country, religion, education, lifestyle, job, etc.
(2) *"What"* represents what a customer has done, that is, their behavior. A behavior is expressed by a predicate and several variables, say, a transaction in a stock market would be *(Sell, stock m, at the price of x, with n shares)*, where the *stock*, *price* and *number of shares* are variables of the behavior *Sell*.
(3) *"When"* represents the time a behavior is performed, or temporary information. For example, when a customer calls and complains or when an insurance policy expires.

Client behavior analysis is usually based on human demographics, financial situation data, transactional data, etc. Further, we are interested in investigating the relationship between client behavior and the business target, i.e., what kinds of taxpayers' payment behavior patterns are likely to be compliant with their debt case.

**Table 1** Client attributes

| Client ID | Customer Attributes | | | | Class |
|---|---|---|---|---|---|
| | Gender | Age | Debt Amount | Client Type | |
| 1001 | M | 20 | 1,032 | Company | Compliant |
| 1002 | F | 50 | 4,665 | Individual | Compliant |
| 1003 | N/A | 45 | 99 | Trust | Non-compliant |
| ... | ... | ... | ... | ... | ... |

In the literature, most client behavior analysis work focuses on *Who*, and partly on *What* and *When*, but doesn't link them together to find combined and more interesting patterns. Only some of the existing work combines pattern mining [9, 27]. However, customers with different demographic characteristics, such as age or gender, may have different behavior patterns. For example, given a behavior pattern, i.e., $< Payment \leq 1,000 \rightarrow AutoAction >$, for customers with the attribute *ClientType=Company*, this may indicate compliance, but for customers with *ClientType=Individual*, it might more likely indicate non-compliance. Therefore, combining client attributes and behavior data will contribute to more interesting patterns.

**Behavior-related variables** Client behavior-related data can be described by the following:

*Client Attributes* are composed of attributes representing a client's general background information, such as demographic data and financial statistical data (see Table 1). Client attribute variables are denoted by $A$, $A = \{a_1, a_2, \cdots a_m | a_i$ *is an client attribute*, $i = 1, \cdots, m\}$.

*Client (Sequential) Behavior* refers to all clients' activities, interactions, transactions and how he/she interacts with the service provider, in time-order. They are denoted by $B=\{b_1, b_2, \cdots, b_n | b_i$ *is a type of behavior*, $i = 1, \cdots, n\}$, such as cash flow transactions, balance enquiry, making a complaint, asking for a special form, changing personal details, changing financial advisors and so on (see Table 2).

*Client Classes* refer to the label/flag of a given client, i.e., if he/she is active or churn, or compliant or non compliant. It is denoted by $C$, i.e., $C = \{c_1 = active, c_2 = churn\}$, $C = \{c_1 = compliant, c_2 = noncompliant\}$.

**Combined customer behavior sequence data** Customer attributes (Table 1) and behavior data (Table 2) can be integrated to become combined data (see Table 3), which we call *Combined Behavior Sequence Data*. The new dataset is represented by sequences.

**Table 2** Client (Sequential) behaviors

| Client ID | Behavior Sequences |
|---|---|
| 1001 | < 1531 →AA→ARR→ 4121 → 4122 > |
| 1002 | <AA→ 4121 > |
| 1003 | <ARR→AA→ARR→ 4121 > |
| ... | < ... →... > |

**Table 3** Combined client behavior sequence data (Table 1 + Table 2)

| | Combined Behavior Sequences | | |
|---|---|---|---|
| Client ID | Client Attributes (Gender, Age, Debt, Client Type) | Behavior Sequences | Class |
| 1001 | $< (M, 20, 1032, Company)$ | $\rightarrow 1531 \rightarrow AA \rightarrow ARR \rightarrow 4121 \rightarrow 4122 >$ | Compliant |
| 1002 | $< (F, 50, 4665, Individual)$ | $\rightarrow AA \rightarrow 4121 >$ | Compliant |
| 1003 | $< (N/A, 45, 99, Trust)$ | $\rightarrow ARR \rightarrow AA \rightarrow ARR \rightarrow 4121 >$ | Noncompliant |
| ... | ... | | |

Each customer has one sequence consisting of customer attributes and sequential behavior. Each attribute is considered an item of a customer sequence (numeric attributes will be discretized). For example, for customer 1001, its combined behavior data is represented by the following sequence:

$$< (Gender = M, Age = 20, DebtAmount = 1,032, ClientType = Company) \rightarrow 1531 \rightarrow AA \rightarrow ARR \rightarrow 4121 \rightarrow 4122 >$$

When client attributes and behavior data are combined, they will contribute to more interesting patterns.

### 3.3 Interesting client behavior CSP

**Contrast client behavior sequential pattern** A $Behavior\ Pattern\ BP$ is composed of a set of client attributes and behavior information, denoted by $\{A, B\}$, where $A$ represents one or more client attributes; and $B$ stands for a set of behaviors. $A$ or $B$ is allowed to be an empty value since some clients might only have basic information or only behavior data. This kind of pattern $BP = \{A, B\}$ not only reveals clients' behavior patterns, but also their attributes. It can help to verify what kinds of clients may have common behaviors.

Therefore, a behavior pattern $X$ is defined as a *Contrast Client Behavior Sequential Pattern*, if:

(1) $sup(X, c_1) \geq min\_sup\_c_1$, where $min\_sup\_c_1$ is a minimum support threshold for class $c_1$; $sup(X, c_2) \geq min\_sup\_c_2$, where $min\_sup\_c_2$ is a minimum support threshold for class $c_2$;
(2) $CR(X) > min\_cr$, where $min\_cr$ is a minimum contrast rate threshold.
(3) $\forall Y \subset X$, if $GR_{c_i}(X) \geq 1$, then $GR_{c_i}(Y) < GR_{c_i}(X)$.

This rule will keep short patterns and delete long patterns if the growth rate of a short pattern is greater than any of its subsequences. For example, given two similar patterns, $X =< abc >$ and $Y =< bc >$, $Y \subset X$, if $GR_{c_1}(Y) > GR_{c_1}(X) \geq 1$, we ignore $X$ and keep $Y$ since $Y$ is a shorter pattern and has a higher contrast rate. In this way, redundant CSPs are removed.

**High-impact behavior factor** In addition to a high contrast rate CSP, we are also interested in finding high impact factors/behaviors.

A *high-impact factor* refers to a special attribute/behavior which can have a reverse effect on a specific pattern. Given a growth rate threshold $min\_cr$ and a pair of CSP $X$ and $Y$, if the following characteristics exist, it is called a *high-impact factor pair*,

1) $Y$ is a subsequence of $X$, for example, $X =< a\ b\ c >$ and $Y =< a\ b >$;
2) $\frac{sup(X,c_2)/|D_2|}{sup(X,c_1)/|D_1|} > min\_cr$; and
3) $\frac{sup(Y,c_1)/|D_1|}{sup(Y,c_2)/|D_2|} > min\_cr$.

and $c$ is the high impact factor for the pattern $Y$.

A high impact pattern pair is rare in a real-life dataset when setting a high threshold of contrast rate $min\_cr$, especially in JCSP. Therefore, a lower growth rate threshold is expected in mining high-impact factor pairs.

## 4 Methodology

### 4.1 CSP-tree

We build a CSP-tree structure, as shown in Figure 2, to mine CSPs.

In the tree, each node is composed of three parts: (a) the prefix in the current path, such as $\{b\}$ in Node N1, as shown in Figure 3; (b) the projected database under the current prefix. We keep the class information, $c_1$ or $c_2$, for each sequence in the projected database; and (c) the frequency/support of each class, i.e. $c_1$ and $c_2$, in its corresponding projected database. We need to know the support of different classes in each tree node, to calculate the contrast rate, therefore, each node in the CSP-tree must record the frequency of the prefix for each class.

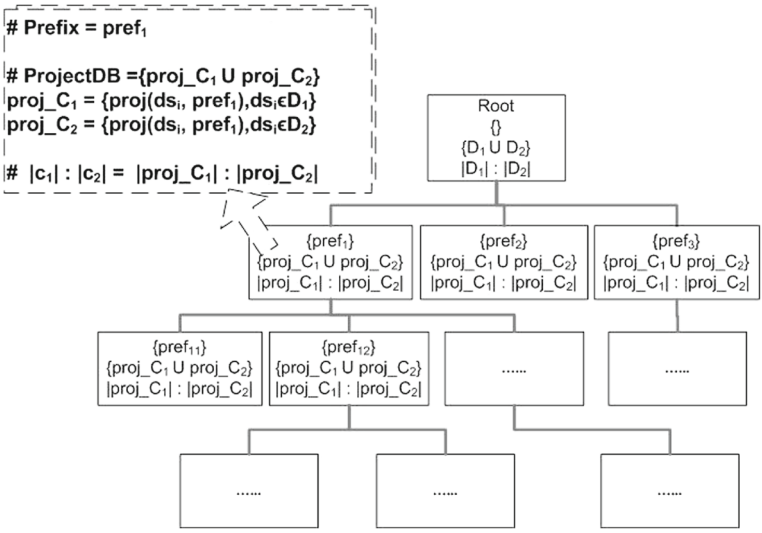



**Figure 2** The Structure of the CSP-tree

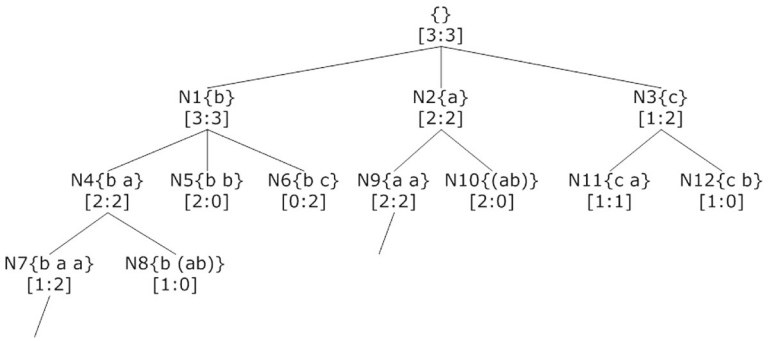



**Figure 3** The CSP-tree of the example dataset

The CSP-tree is based on PrefixSpan [22] tree. Using the CSP-tree structure results in four possible advantages:

(1) No candidate sequence needs to be generated, and projected databases keep shrinking [22]. It does not generate or test any candidate sequence which is nonexistent in a projected database. The number of sequences in a projected database will become much smaller when the prefix grows [22].
(2) The compressed sequence database is allowed to be kept in memory.
(3) It is easy to count the frequency of different classes so calculating the contrast rate of potential CSPs becomes possible.
(4) It is feasible for Apriori-like pruning and some support-based pruning measures.

### 4.2 eCSP Algorithm

This section details an effective CSP mining algorithm, called eCSP, which is based on a CSP-tree structure. Generally, it implements a downward and depth-first search strategy on the CSP-tree to traverse all potential CSPs, and uses some pruning measures to reduce the search space. For each node in the tree, its prefix and the corresponding projected database are examined first, then CSPs are generated by counting the frequency of different classes in the projected database.

Step 1: Find all frequent 1-item patterns, which have greater support than a predefined support threshold $min\_sup$, make them prefixes, and then obtain their corresponding projected databases. For example, $\{a\}$, $\{b\}$ and $\{c\}$, in Figure 2, are frequent 1-item patterns and each of them has a corresponding projected database.

Step 2: For the 1-item patterns, do depth-first processing until: (1) no more frequent 1-item patterns can be found in the corresponding projected database; (2) or any support of the two classes equals 0. For example, as illustrated in Figure 2, the algorithm scans the projected database of $\{a\}$ to obtain its frequent 1-item patterns first, such as $\{a\ a\}$, $\{a\ b\}$ and $\{a\ c\}$, and the corresponding projected databases. Then, starting from node $\{a\ a\}$, if $sup(<\ a\ a\ >, c_1) = 0$ or $sup(<\ a\ a\ >, c_2) = 0$, it stops here and goes back to its parent node; otherwise, it goes deeper and gets two child nodes $\{a\ a\ a\}$ and $\{a\ a\ b\}$. It keeps going on until no more frequent 1-item patterns can be found in the corresponding projected databases.

Step 3: If all child nodes have been traversed, it processes the next sibling node until all nodes are traversed.

**Pseudocode** The pseudocode of the approach is given as Algorithm 1 and Algorithm 2.

**Algorithm 1** RunCSP

```
Input: SequenceDatabase database, min_cr, max_length
Map<Prefix,Set<Integer>> mapSeqID=findFrequentItems(database);
for Entry entry : mapSeqID.entrySet() do
    if entry.getValue().size() >= min_sup then
        Prefix pref = entry.getKey();
        projectDB = buildProjectDB(pref, database);
        if (CR(pref)>min_cr) then
            //found and output a CSP
    if pref.sup1==0 || pref.sup2==0 then
        continue; //don't need go deeper
    Recursion(pref, 2, projDB);
```

**Algorithm 2** Recursion

```
Input: pref, level, projectDB
Set<Pair> pairs=findAllFrequentPairs(prefix, projectDB);
for Pair p : pairs do
    Prefix pref2 = getNewPrefix(pref, p.pref);
    ProjectDB pDB2=buildProjectDB(p.item, projectDB, p.isPostfix);
    if CR(pref)>min_cr then
        //found and output a CSP;
    if p.sup1==0||p.sup2==0 then
        continue;
    if Prune==true then
        continue;
    if k<max_len then
        Recursion(pref2, level + 1, pDB2);
```

**An example** Here we give a simple example to explain the approach. Table 4 is an example dataset. Given $min\_cr = 2$ (we set $min\_sup = 3$ in the example to make it easy to understand), Figure 3 shows the CSP-tree structure of the dataset, and Table 5 shows the potential CSPs and projected databases for each step.

**Table 4** An example data set

| ID | Data Sequence | Class |
|---|---|---|
| 1 | $< b\ (ab)a >$ | $c_1$ |
| 2 | $< b\ b\ d >$ | $c_1$ |
| 3 | $< c\ (ab)a >$ | $c_1$ |
| 4 | $< b\ c >$ | $c_2$ |
| 5 | $< b\ a\ a >$ | $c_2$ |
| 6 | $< b\ c\ a\ a >$ | $c_2$ |

**Table 5** Projected databases and CSP

| Node | Prefix | Projected Databases | $c_1 : c_2$ | CR |
|---|---|---|---|---|
| N1 | $\{b\}$ | 1. $< (ab)\ a >$ $[c_1]$; 2. $< b >$ $[c_1]$; 3. $< a >$ $[c_1]$; 4. $< c >$ $[c_2]$; 5. $< a\ a >$ $[c_2]$; 6. $< c\ a\ a >$ $[c_2]$ | 3:3 | 1 |
| N2 | $\{a\}$ | 1. $< (\_b)\ a >$ $[c_1]$; 3. $< (\_b)\ a >$ $[c_1]$; 5. $< a >$ $[c_2]$; 6. $< a >$ $[c_2]$ | 2:2 | 1 |
| N3 | $\{c\}$ | 3. $< (ab)\ a >$ $[c_1]$; 4. $<>$ $[c_2]$; 6. $< a\ a >$ $[c_2]$ | 1:2 | 2 |
| N4 | $\{b\ a\}$ | 1. $< (\_b)\ a >$ $[c_1]$; 3. $<>$ $[c_1]$; 5. $< a >$ $[c_2]$; 6. $< a >$ $[c_2]$ | 2:2 | 1 |
| N5 | $\{b\ b\}$ | 1. $< a >$ $[c_1]$; 2. $<>$ $[c_1]$; | 2:0 | $\infty$ |
| N6 | $\{b\ c\}$ | 4. $<>$ $[c_2]$; 6. $< a\ a >$ $[c_2]$; | 0:2 | $\infty$ |
| N7 | $\{b\ a\ a\}$ | 1. $<>$ $[c_1]$; 5. $<>$ $[c_2]$; 6. $<>$ $[c_2]$ | 1:2 | 2 |
| N8 | $\{b\ (ab)\}$ | 1. $< a >$ $[c_1]$; | 1:0 | $\infty$ |
| N9 | $\{a\ a\}$ | 1. $<>$ $[c_1]$; 3. $<>$ $[c_1]$; 5. $<>$ $[c_2]$; 6. $<>$ $[c_2]$ | 2:2 | 1 |
| N10 | $\{(ab)\}$ | 1. $< a >$ $[c_1]$; 3. $< a >$ $[c_1]$ | 2:0 | $\infty$ |
| N11 | $\{c\ a\}$ | 3. $< (\_b)\ a >$ $[c_1]$; 6. $< a >$ $[c_2]$ | 1:1 | 1 |
| N12 | $\{c\ b\}$ | 3. $< a >$ $[c_1]$ | 1:0 | $\infty$ |

In the first step, all 1-item prefixes, i.e. $\{b\}$, $\{a\}$ and $\{c\}$, were found since their supports are not less than $min\_sup = 3$. Since the support of $d$ is only 1, which is less than $min\_sup$, it is ignored in the CSP-tree. Nodes N1, N2 and N3 are built for the three 1-item prefixes, and the project databases are created for the 3 nodes as well, as shown in Table 5.

Then, for the prefix $\{b\}$, since $sup(< b >, c_1)$ and $sup(< b >, c_2)$ are more than zero, we go deeper to search for its child nodes. Based on the projected database of $\{b\}$, three frequent items, $\{a\}$,{b} and $\{c\}$, are found, and three nodes, N4, N5 and N6, are created, respectively (see Table 5).

Again, while going deeper from node N4, we get N7 and N8. Since N7 doesn't have any child nodes and its contrast rate is 2, the corresponding pattern $< b\ a\ a >$ is identified as a CSP. Then, we check N8 and find its support for $c_2$ is 0, which means $< b\ (ab) >$ is a JCSP, so we won't go deeper for the node. Therefore, we return to its parent level to check the parent node's sibling node N5. N5 $< b\ b >$ also turns out to be a JCSP, as is N6 as a JCSP as well.

We now have four CSPs and three are JCSP. As there are no other child nodes to be checked, we go back to the parent node N1, and check its next sibling node N2 to traverse its child nodes, and N9 and N10 are found. Since N9 doesn't have any child nodes and its contrast rate is 1, it is not a CSP. N10 is a CSP since its support for $c_2$ is 0, and 2 for $c_1$. After N10, there are no more nodes on the same level, so it traverses back to check its parent node N3. Then the procedure continues for every node under N3 until there are no more nodes being generated.

### 4.3 Pruning methods

Since the Apriori property can't be employed in CSP pruning, it is hard to prune unnecessary nodes efficiently. So far, we have not found any effective pruning methods that keeps all the results without any loss. Hence, we adopt a Chi-square heuristic pruning method to find the CSPs with high discriminative characteristics. This can improve the algorithm's computing

performance significantly which enables us to focus on more interesting patterns. Several other measures are introduced in this section.

**Chi-square pruning** This measure compares the sample distribution between a parent node $X$ and its child node $Y$.

If $a = sup(X, c_1)$, $b = sup(X, c_2)$, $c = sup(Y, c_1)$, $d = sup(Y, c_2)$,

$$chi(X, Y) = (a + b + c + d) * (ad - bc)^2/[(a + b)(c + d)(a + c)(b + d)] \quad (5)$$

If $chi(X, Y)$ is less than a predefined value $\xi$ (i.e. 3.84), then node $Y$ will be pruned. Usually, we set the threshold at 3.84 since it is the chi-square statistic value for a test with 1 df (degree of freedom). It is capable of identifying highly interesting CSPs and deleting some redundant ones when there are lots of patterns.

**Pruning Low Support Nodes** There are two strategies for pruning low-support nodes.

(1) The first targets all groups/datasets. If the total support of all classes in a given node is lower than $min\_sup$, then the node will be ignored. This method is popular and has been adopted in frequent pattern mining. In CP mining, it is still capable of pruning low support patterns [5, 10, 24].

(2) The second targets only one class if this is the area of focus. For example, we set a threshold $min\_sup\_c_1$ for $c_1$ and $min\_sup\_c_2$ for $c_2$. Given a node $X$ with $m$ $c_1$ class and $n$ $c_2$ class, if $m < min\_sup\_c_1$, the node will be ignored. The measure is applicable when we are interested in a special class, for example, in fraud detection, we might be interested in fraud records and set a threshold for fraud, rather than genuine records.

**Pruning based on business significance** Although support is a vital measurement, sometimes we are interested in business values and significance, for example, balance, client benefit, profit and so on. If the sum values, i.e., profit, of a special node are low, we can prune it since the group has a low value for business purposes. The sum value of each node in the CSP-tree will follow a convergence property, so that it is efficient for pruning low value CSPs.

Another effective pruning measure is to set the maximum length of a pattern. Since longer patterns are harder to explain using business knowledge, we can set a maximum length threshold to reduce the search space much more efficiently.

## 5 Evaluation

We tested the proposed approach on the following three real-world datasets.

Dataset 1 (*DS*1) is from the financial service industry. It is composed of demographic data, product-related data and behavior data of 50,000 records/sequences; half of them represent churn clients, the other half are active clients. The average number of elements in a sequence is 85.

Dataset 2 (*DS*2) is composed of the demographic data and behavior data of 4,690 customers from insurance claim data, and each client has 6 attributes and a behavior sequence. The average number of elements in a sequence is 21, while the maximum number is 144.

Dataset 3 (*DS*3) is a sample dataset from the tax debt collection area. It is composed of demographic data, product-related features and taxpayers' behavior data, including lodgment, payment or action transactions. We de-identified the dataset in terms of client id,

and even the behavior item code. It contains 41,212 records/sequences, around half of them are self-finalised, the others are non-self-finalised. The average number of elements in a sequence is 15.

The proposed approach is implemented by Java, and the following experiments are conducted on a PC with Intel Core 2 CPU of 2.66 GHz, 4 GB memory and Windows XP system with SP3.

### 5.1 Performance of pruning methods

Figure 4 shows a comparison between the Naive method, which is without any pruning, and a heuristic chi-square pruning measure, as described in Section 4.3. In the experiment, we set $min\_cr = 5$ on both $DS1$ and $DS2$, and set $min\_cr = 3$ for $DS3$. ($min\_cr$ is the minimum contrast rate to define the interestingness of the contrast patterns. A higher rate represents higher discriminative patterns. Usually, we set it between 3 and 10, it won't have much impact on the execution time, only on the final patterns we output.) The line for "Naive" in Figure 4 stands for the Naive method, which is without any pruning, and the line for "Chi-square" represents the Chi-square pruning measure. The results show that the Naive method is very time-consuming even though it can find all the CSPs, and Chi-square pruning can only find a small part of the most interesting CSPs in a much shorter execution time than the Naive method. This is a good trade-off in the real-world application.

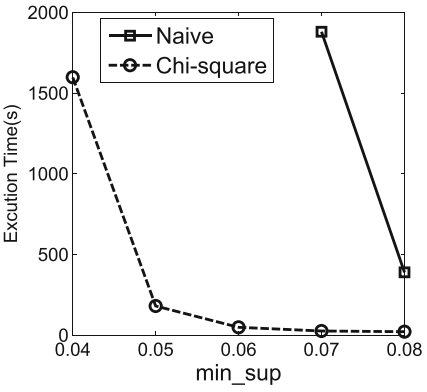
### 5.2 Scalability test

A scalability test was also conducted to determine the robustness of the proposed algorithm on large datasets. Figure 5 shows the results of the three datasets $DS1$, $DS2$ and $DS3$, of different sizes, which are 1 to 4 times larger than the original datasets, in terms of the number of data sequences. In the experiment, we set $min\_sup = 0.05$ on $DS1$, $min\_sup = 0.0066$ on $DS2$, and set $min\_sup = 0.0024$ for $DS3$. ($min\_sup$ is used to control the number of candidates so as to keep the maximum execution time at a reasonable level. Since a very low $min\_sup$ might generate a huge amount of candidates and high-complexity, it might take a few days or even be impossible to finish. Therefore, we set the $min\_sup$ at a level which could limit the execution time to around 2000 seconds or so. That is why we set different $min\_sup$ thresholds for the three datasets.)

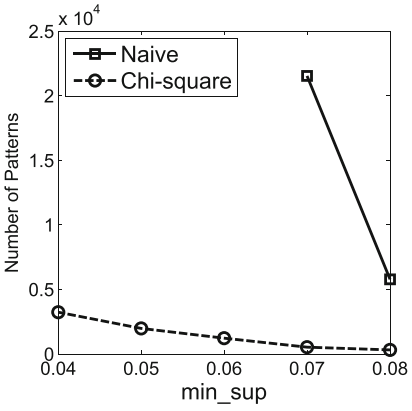
For $DS1$, we tested 50 k, 100 K, 150 K and 200 K datasets. When the data size increased to 200 K on $DS1$, it took 732 seconds to obtain the results. This is around four times the execution time when the sample data size is 50k, which is 180 seconds. Therefore, a four-fold increase in data size, result in a four-fold increase in execution time. Therefore, the execution time increases linearly with the number of records. A similar relationship is evident in the other two datasets. The results show that the computing performance of the proposed method is robust on large datasets.

### 5.3 Case study and business evaluation

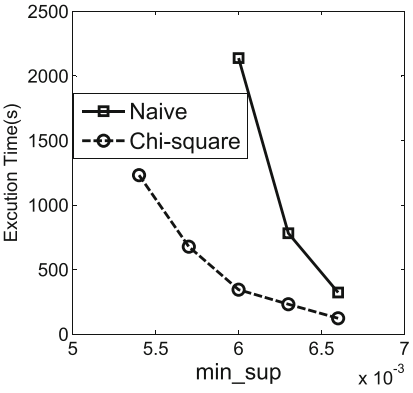
The case study targets tax debt collection optimization by using taxpayers' static (attributes) and sequence data (lodgment/payment/interaction transactions) to find which behavior patterns are likely to result in self-finalised($s$)/compliant debt cases (which don't need any action to be taken to collect, so as to save resources), and which patterns are likely to be non-self-finalised ($n$)/non-compliant (which require early action for earlier collection).



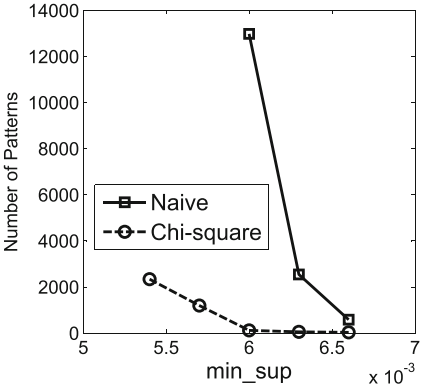

(a) Execution Time (DS1)



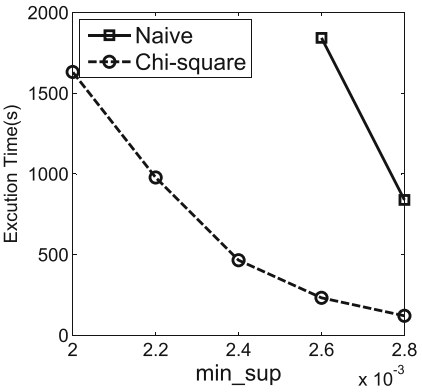

(b) Number of CSPs (DS1)



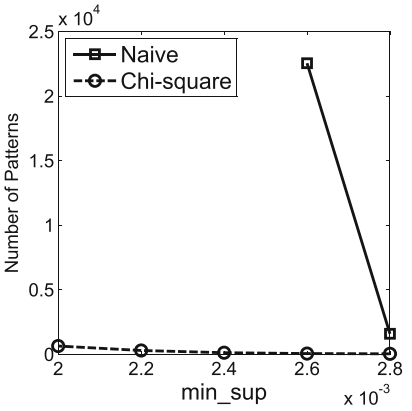

(c) Execution Time (DS2)




(d) Number of CSPs (DS2)



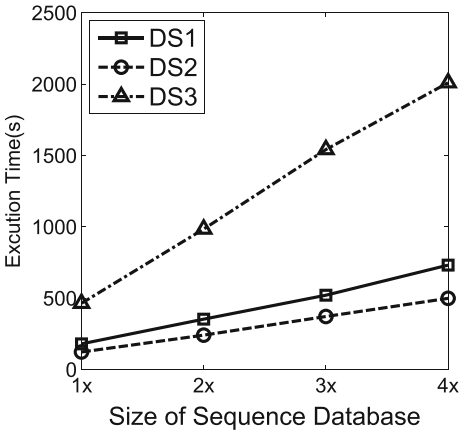

(e) Execution Time (DS3)




(f) Number of CSPs (DS3)

**Figure 4** Performance comparison

**Figure 5** Scalability test




We focus on the following aims to evaluate the proposed approach in this paper: (1) to find interesting business rules to identify which kinds of taxpayers/cases will be self-finalised ($s$) or non-self-finalised ($n$); and (2) to find which attributes or behaviors which could be high-impact factors, which can cause taxpayers to change from highly non-self-finalised to highly self-finalised, and vice versa.

**Mining discriminative business rules** CSP is very useful for mining discriminative sequential patterns, which can become valuable business rules to verify what kinds of customers or what kinds of behaviors may result in self-finalised cases. Such kinds of patterns, especially JCSP, are very helpful for predicting which cases are most likely to self-finalise.

Here, we list some example CSPs in Table 6. These rules come from dataset *DS*3 by setting $min\_sup = 0.0022$ and $min\_cr = 3.0$. *Rule* 1 means that there are 1446(= 1099 + 347) clients/cases who have paid more than twice in the last 60 days, and have a payment behavior with PaymentCode=4627 before a case creation, 1099 of them are self-finalised(s) and 347 of them are non-self-finalised(n). The comparison rate is 3.2, which shows the population has a much higher probability of being self-finalised. *Rule* 8 means that the *Individual* clients who have lodged an activity statement once in the last 60 days, and the

**Table 6** Examples of discriminative business rules

| ID | Example Rules | s | n | $GR_s$ | $GR_n$ |
|---|---|---|---|---|---|
| 1 | <(Payment_Count_InLast60Days>= 2) → PaymentCode=4627> | 1099 | 347 | 3.2 | - |
| 2 | <(Payment_Count_InLast60Days>= 2) → PaymentCode=4622> | 1211 | 394 | 3.1 | - |
| 3 | <(Payment_Count_InLast60Days>= 2) → PaymentCode=1531 → PaymentCode=4627> | 492 | 156 | 3.2 | - |
| 4 | <(Payment_Amount_InLast60Days=[-125972.0, -73003.12)) > | 229 | 76 | 3.0 | - |
| 5 | <(Client_Type = Partnership, Has_ABN = No) → Auto Action > | 0 | 162 | - | $\infty$ |
| 6 | <(Client_Type = Partnership, Has_ABN = No) → KIM Action> | 0 | 175 | - | $\infty$ |
| 7 | <(Client_Type = Individual, Has_ABN = No) > | 1 | 178 | - | 178 |
| 8 | <(Client_Type = Individual, AS_Num_InLast60Days<=1, NumOfDays_From_Last_Activity = [47, 52)) > | 404 | 1365 | - | 3.4 |

**Table 7** Some examples of high impact factor pairs

| ID | Example of High-impact Factor Pairs | $GR_s$ | $GR_n$ |
|---|---|---|---|
| 1 | < (AS_Amount_InLast60Days=[0.0, 7844.0), MarketSegment = Micro, NumOfActivity=[1,2) ) > | 3.0 | - |
| 1 | < (AS_Amount_InLast60Days=[0.0, 7844.0), MarketSegment=Micro, NumOfActivity=[0,1) ) > | - | 5.9 |
| 2 | < (Credit_InLast60Days≥0, MarketSegment=Micro) → Auto Action > | 3.0 | - |
| 2 | < (Credit_InLast60Days≥0, MarketSegment=Micro) → KIM Action > | - | 3.6 |
| 3 | < (Credit_InLast60Days≥0, IncomeYear_Diff=[1.0,2.0)) → KIM Action > | 3.0 | - |
| 3 | < (Credit_InLast60Days≥0, IncomeYear_Diff=[1.0,2.0)) → Negotiation Action > | - | 3.2 |

number of days since their last activity is between 47 and 52, then 404 of such cases are self-finalised(s) and 1365 of them are non-self-finalised(n). The comparison rate is 3.4, which shows the population has a much higher probability of being non-self-finalised. *Rule* 5 and 6 are two JCSPs since their comparison rates are equal to infinity.

These rules help business people to understand client behaviour, and they show the potential benefits of customer segmentation. We investigated the top 20 self-finalised and non-self-finalised rules with business experts, and most were convinced and represent various business scenerios/stories, and so far not within the existing knowledge.

**Mining high impact factors/behaviors** As described in Section 3.3, given two CSPs $X$ and $Y$, if $X$ is highly related to self-finalised(s) cases and CSP $Y$ is highly related to non-self-finalised(n) ones, and $f$ denotes a behavior/factor which has a reversing impact on $Y$, we call it a high impact factor for the CSP pair $X$ and $Y$.

Several examples of pattern pairs are listed in Table 7. In the first pairs, *Activity* is a high-impact factor for the rule $(AS\_Amount\_InLast60Days = [0.0, 7844.0), MarketSegment = Micro)$, that is to say, if $NumOfActivity = 1$, the cases are likely to be self-finalised ($GR_s = 3.0$), but if $NumOfActivity = 0$, it will be much more likely to be non-self-finalised ($GR_n = 5.9$). Different activity numbers might have different impacts.

The second pair shows that two action behaviors, Auto Action or KIM action, are high impact factor of rule $(Credit\_InLast60Days \geq 0, MarketSegment = Micro)$. The cases which have Auto action are likely to be self-finalised, but the ones with KIM action are likely to be non-self-finalised. These rule pairs indicates the high impact factors in the comparison. Such pairs and high-impact factors are valuable and helpful for business, for example, exploring how to change behaviour, especially ATO-sponsored behaviour, i.e. outbound call/letter, so as to trigger the impact change, to encourage more people to change from the non-self-finalised group to the self-finalised group.

## 6 Conclusions and future work

In this paper, we proposed an effective CSP mining approach on client sequential behavior analysis. The proposed approach is called eCSP, and utilizes a data structure of a CSP-tree to traverse all potential CSPs. A few effective pruning methods were proposed as well, to reduce the search space significantly. Some experiments were carried out on three real-world datasets to evaluate the pruning measures and the approach's scalability. We used the

model to solve a real business problem in the debt collection area, and the output shows promising and convincing value for business.

Further work will focus on more effective pruning measures to find interesting CSPs, and will build an effective classifier based on the discovered CSPs. Other work will undertake contrast pattern evaluation, in terms of pattern coverage and overlap, to evaluate how many interesting patterns we have and how many are lost.

## References


1. Agichtein, E., Zheng, Z.: Identifying best bet web search results by mining past user behavior. In: KDD 2006, 902–908. ACM (2006)
2. Agrawal, R., Srikant, R.: Mining sequential patterns. In: ICDE, pp. 3–14 (1995)
3. Attenberg, J., Pandey, S., Suel, T.: Modeling and predicting user behavior in sponsored search. In: KDD 2009, pp. 1067–1076, ACM. (2009)
4. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential Pattern Mining Using a Bitmap representation. In: KDD 2002, pp. 429–435 (2002)
5. Bailey, J., Manoukian, T., Ramamohanarao, K.: Fast algorithms for mining emerging patterns. Prin Data Min. Knowl. Disc. **2431**, 187–208 (2002)
6. Bayardo, R.J.: Efficiently Mining Long Patterns from Databases. SIGMOD (1998)
7. Chan, S., Kao, B., Yip, C., Tang, M.: Mining emerging substrings. In: DASFAA 2003, pp. 119–126 (2003)
8. Cao, L.: Behavior informatics and analytics: Let behavior talk. In: ICDM 2008 Workshops, pp. 87–96 (2008)
9. Cao, L., Zhang, H., Zhao, Y., Luo, D., Zhang, C.: Combined mining: Discovering informative knowledge in complex data. IEEE Trans. Syst. Man. Cybern. B. Cybern. **41**(3), 699–712 (2011)
10. Dong, G., Li, J.: Efficient mining of emerging patterns: discovering trends and differences. In: KDD 1999, pp. 43–52 (1999)
11. Dong, G., Li, J., Zhang, X.: Discovering Jumping Emerging Patterns and Experiments on Real Datasets. (IDC99) (1999)
12. Dong, G., Zhang, X., Wong, L., Caep, J.Li.: Classification by aggregating emerging patterns. In: Discovery Science, vol. 1721, pp. 737–737 (1999)
13. Fan, H., Ramamohanarao, K.: Efficiently mining interesting emerging patterns. In: WAIM2003, pp. 189–201 (2003)
14. Fan, H., Ramamohanarao, K.: Fast discovery and the generalization of strong jumping emerging patterns for building compact and accurate classifiers. TKDE **18**(6), 721–737 (2006)
15. Han, J., Pei, J., mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M.-C.: Freespan: Frequent Pattern-projected Sequential Pattern Mining. In: KDD, pp. 355–359 (2000)
16. Ji, X., Bailey, J., Dong, G.: Mining minimal distinguishing subsequence patterns with gap constraints. Knowl. Inf. Syst. **11**, 259–286 (2007)
17. Li, W., Han, J., Pei, J.: CMAR: Accurate and efficient classification based on multiple class-association rules. In: ICDM 2001, pp. 369-376 (2001)
18. Loekito, E., Bailey, J.: Fast mining of high dimensional expressive contrast patterns using binary decision diagrams. In: SIGKDD 2006, pp. 307–316 (2006)
19. Mannila, H., Toivonen, H.: Levelwise Search and Borders of Theories in Knowledge Discovery. Data Min. Knowl. Disc. **1**(3), 41 (1997)
20. Mozer, M., Wolniewicz, R., Grimes, D., Johnson, E., Kaushansky, H.: Predicting subscriber dissatisfaction and improving retention in the wireless telecommunica- tions industry. IEEE Trans. Neural Netw. **11**(3), 690–696 (2000)
21. Pasquier, N., Bastide, R., Taouil, R., Lakhal, L.: Efficient Mining of Association Rules using Closed Itemset Lattices. Information Systems 24(1) (1999)
22. Pei, J., Han, J., Asl, M.B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: PrefixSpan Mining Sequential Patterns Efficiently by Prefix Projected Pattern Growth. In: ICDE, pp. 215–226 (2001)
23. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, Los Altos (1993)

24. Ramamohanarao, K., Bailey, J.: Emerging patterns: mining and applications. In: ICISIP 2004, pp. 409–414 (2004)
25. Wang, X., Duan, L., Dong, G., Yu, Z., Tang, C.: Efficient Mining of Density-Aware Distinguishing Sequential Patterns with Gap Constraints. DASFAA 372–387 (2014)
26. Zaki, M.J.: SPADE: An efficient algorithm for mining frequent sequence. Mach. Learn. **42**, 31–60 (2001)
27. Zhao, Y., Zhang, H., Cao, L., Zhang, C., Bohlscheid, H.: Combined Pattern Mining: From Learned Rules to Actionable Knowledge. AI 393–403 (2008)