THE NATIONAL UNIVERSITY
of SINGAPORE


School of Computing
Lower Kent Ridge Road, Singapore 119260

**TRA6/06**

# *Mining Progressive Confident Rules*

*Minghua ZHANG, Wynne HSU and Mong Li LEE*

*June 2006*

# Technical Report

## Foreword

*This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.*

JAFFAR, Joxan
Dean of School

# Mining Progressive Confident Rules

Minghua Zhang Wynne Hsu Mong Li Lee

School of Computing,
National University of Singapore.
{*zhangmh, whsu, leeml*}*@comp.nus.edu.sg*

## ABSTRACT

Many real world objects have states that change over time. By tracking the state sequences of these objects, we can study their behavior and take preventive measures before they reach some undesirable states. In this paper, we propose a new kind of pattern, called progressive confident rules, to describe sequences of states with an increasing confidence that lead to a particular end state. We give a formal definition of progressive confident rules and their concise set. We propose new pruning strategies and employ the concise set analysis of rules in the mining process to reduce the enormous search space. Experiment result shows that the proposed algorithm is efficient and scalable. We also demonstrate the application of progressive confident rules in classification.

## 1. INTRODUCTION

Real life objects can be described by its attribute values. For example, a person has attributes such as gender, date of birth, education level, and job, etc. While the gender and date of birth of a person do not change, the education level and job may change with time. If we denote the set of attribute values of an object as its state, then the state of an object changes as the attribute values change with time. The states of an object at different time stamps form a state sequence.

In many applications, an object's state sequence over time is usually more interesting than its current state because the state sequence depicts the object's behavior characteristics. For example, if we only look at the current stock price, we cannot tell its behavior in the next week. However, if we look at its price history over several months, we could have a better idea about its trend in the future.

The state sequence of an object also captures more information than the current state for classification. For example, it is hard to determine whether a patient has chronic lymphocytic leukemia if we only knows that he currently has anemia. However, if we track the patient's medical history over a period of time, then the doctor will be able to make a better judgement.

Table 1 shows a sample database that records the symptom sequences of patients and whether they have Chronic Lymphocytic Leukemia (CLL). The first column is the patient's ID. The second column is the patient's symptom sequence over time. For example, for patient ID 1, he/she first has night sweat and hypodynamia, followed by fever, then achroacytosis, and finally anemia. We regard the set of symptoms at a time point as the state of a patient at that time. Thus, the first state of patient 1 is {night sweat, hypodynamia}, and his last state is anemia. The last column in Table 1 is the doctor's diagnosis on Chronic Lymphocytic Leukemia(CLL). The first four patients have CLL, while the last four do not.

| ID | Symptom Sequence | CLL |
|---|---|---|
| 1 | {Night sweat, hypodynamia } →Fever →Achroacytosis→Anemia | Y |
| 2 | Night sweat→Fever→Achroacytosis→Anemia | Y |
| 3 | Night sweat→Fever→Achroacytosis→Anemia | Y |
| 4 | Night sweat→Fever→Achroacytosis→Splenomegalia | Y |
| 5 | Night sweat→Fever→Achroacytosis | N |
| 6 | Night sweat→Fever→Anemia | N |
| 7 | Night sweat→Splenomegalia→Anemia | N |
| 8 | Night sweat→Sleepy→Anemia | N |

**Table 1: Example Diagnosis on Chronic Lymphocytic Leukemia**

If we examine the last state of each patient, or their last symptom, 3 people with anemia have CLL (IDs 1–3), while the other 3 with anemia do not (IDs 6–8). Thus it is not clear whether a patient with anemia will have CLL. However, if we study the patients' state sequences, or their sequence of symptoms, we see that most CLL patients (IDs 1–3) have $night\ sweat \rightarrow fever \rightarrow achroacytosis \rightarrow anemia$. This symptom sequence does not occur in non-CLL patients.

We observe that patient 5 in Table 1 does not have CLL although his symptom sequence is the same as the 3 leading symptoms of CLL patients ($night\ sweat \rightarrow fever \rightarrow achroacytosis$). It is possible that he may have CLL later, and a doctor could advise on preventive measures in advance. Let us call the symptom sequence $night\ sweat \rightarrow fever \rightarrow achroacytosis \rightarrow anemia$ as $p$, and show how $p$ could play a role in predicting patients getting CLL.

Based on Table 1, 4 patients (IDs 1–4) with symptom night sweat have CLL. The total number of patients with night sweat is 8. We use $conf(night\ sweat, CLL)$ to denote the probability of symptom night sweat leading to CLL, that is,

$conf(night\ sweat, CLL) = 4/8 = 0.5$

Similarly, we have

$conf(night\ sweat \rightarrow fever, CLL) = 4/6 = 0.67$

$conf(night\ sweat \rightarrow fever \rightarrow achroacytosis, CLL) = 0.8$

$conf(p, CLL) = 1$

From the four confident values, we see that the probability of a

.

patient getting CLL becomes higher and higher as his symptoms change according to the symptom sequence $p$.

Suppose a new patient comes in with symptom $night\ sweat$. From $conf(night\ sweat, CLL)$, the doctor knows the current probability of the patient catching CLL is not high (0.5), but could monitor whether the next symptoms in $p$ appear later. If fever→achroacytosis occurs, as in the case of patient 5 in Table 1, the doctor could advise on preventive measures because the chance of having CLL has increased (0.8). However, if the patient becomes sleepy, then according to Table 1, we have $conf(night\ sweat \rightarrow sleepy, CLL) = 0/1 = 0 < conf(night\ sweat, CLL)$, and the chance of having CLL is very small.

In this paper, we propose a new kind of pattern that describes the changing states of objects with increasing probabilities that lead to some end state $C$ (e.g. CLL in Table 1). We call the pattern *progressive confident rules*. The general form of a progressive confident rule is

$$X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_n,$$

where $X_i$ is a state, and a state in the left hand side of "→" occurs earlier in time than a state in the right hand side of "→". By progressive confident, we mean the probability of an object achieving state $C$ in the future becomes higher and higher as state $X_i (1 \leq i \leq n)$ appears one after another. We call this the *progressive confident condition*.

For the rules to be more meaningful, we have three additional requirements. First, there is a minimum support of the rule to ensure that the rule is general. Second, the probability of the first state $X_1$ leading to the ending state $C$ should be no less than some threshold $min\_conf1$. This removes rules that begin with unrelated states. For example, if we know that cataract has nothing to do with CLL, we will not be interested in a rule like $cataract \rightarrow night\ sweat \rightarrow fever \rightarrow achroacytosis \rightarrow anemia$ for CLL. Third, the probability of the entire rule leading to $C$ should be no less than a threshold $min\_conf2$. This ensures that the rule is interesting. For example, people do not regard night sweat → fever as a rule for CLL, because many diseases also have this symptom sequence.

We note that the set of all progressive confident rules usually contains redundant information. For example, it is possible that both $night\ sweat \rightarrow fever \rightarrow achroacytosis \rightarrow anemia$ and $night\ sweat \rightarrow fever \rightarrow achroacytosis$ are progressive confident rules. In fact, the former rule subsumes the latter one. We will examine how to discover the concise set of progressive confident rules that does not contain redundant information.

As in most mining problems, the search space for finding progressive confident rules is enormous. Many algorithms use the *apriori* property [14] to prune the search space. The apriori property states that if a pattern satisfies the mining requirement, so do all its sub-patterns. However, this property does not hold for progressive confident rules because we consider the confidence of the rules. Although the support of a pattern is no larger than the support of its sub-patterns, this is not true for confidence.

Another property for pruning the search space is the *prefix anti-monotonic property* [12]. This property states that if a pattern $P$ is in the mining result, so is any pattern $Q$ obtained by removing some ending items from $P$. However, this property does not hold for the progressive confident condition either. By removing some ending items, an itemset may change to its subset, and the confidence of a subset is not necessarily no less than its super-set. Hence, previous pruning methods are not applicable to our problem. We need to devise new pruning strategies. We design an efficient *depth-first* mining algorithm based on the new strategy.

Finally, we show how progressive confident rules can be utilized to predict an object's future state. This is a classification problem in essence. We incorporate the rules into three representative classifiers C4.5 [13], SVM [4] and Bayes Classifier [6] to improve their classification accuracy.

The major contributions of this paper is as follows.

- We introduce a new kind of pattern called progressive rule that intuitively captures the state change of objects leading to some end state with increasing confidence. We find a concise set of rules to represent all the progressive confident rules.
- We devise new pruning strategies for the large search space and describe a depth-first mining algorithm based on these pruning strategies. The algorithm also utilizes the concise set analysis in the mining process to further reduce the huge search space. Experimental results indicate the efficiency and scalability of the algorithm.
- We demonstrate how progressive confident rules can be incorporated into existing classifiers to predict an object's future state. Experiments on both synthetic and real data show that the rules greatly improve the classification accuracy.

The rest of the paper is organized as follows. Section 2 gives a formal definition of progressive confident rules and their concise set. Section 3 describes our algorithm for mining progressive confident rules. Section 4 presents the results of experiments to evaluate the performance of the algorithm. Section 5 compares the classification accuracy of existing classifiers with and without the utilization of progressive confident rules. The related works are reviewed in Section 6, and we conclude in Section 7.

## 2. PROBLEM DEFINITION

In this section, we give a formal definition of progressive confident rules and the notion of a concise set of rules.

### 2.1 Progressive Confident Rules (PCR)

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals called items.

A state or an itemset $X \subseteq I$ is a set of items. The semantic meaning of item and itemset is as follows. Every item is related with a characteristic. If a state (an itemset) contains an item, it means the item's corresponding characteristic appears in the state. For example, if item $a$ represents the symptom "sick", item $b$ represents the symptom "fever", and item $c$ represents the symptom "headache", then state $\{a, b\}$ means the patient feels sick and has a fever but does not have a headache, while state $\{b, c\}$ means the patient has a fever and headache but does not feel sick.

A pattern $P = X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_n$ is an ordered set of states. The length of $P$ is defined as the number of states in $P$. We use $|P|$ to represent the length of $P$. For example, if $P = \{a\} \rightarrow \{b, c\} \rightarrow \{d\}$, then $|P| = 3$.

An observation $o$ is an itemset $T$ with a time stamp ($tid$), i.e., $o = \langle tid, T \rangle$. If a state $X \subseteq T$, we say $X$ is contained in the observation $o$. Sometimes we omit the time stamp $tid$ and represent an observation by its itemset $T$ only.

A sequence $s$ is composed of a sequence id ($sid$) and an ordered list of observations, i.e. $s = \langle sid, o_1, o_2, \ldots, o_n \rangle$.

Given a sequence $s = \langle sid, o_1, o_2, \ldots, o_n \rangle$ and a pattern $P = X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_m$, we say that $s$ matches $P$, or equivalently, $P$ is contained in $s$, if there exist integers $j_1, j_2, \ldots, j_m$, such that $1 \leq j_1 < j_2 < \ldots < j_m \leq n$ and $X_1$ is contained in $o_{j_1}$, $X_2$ is contained in $o_{j_2}$, …, $X_m$ is contained in $o_{j_m}$. We represent this relationship by $P \sqsubseteq s$.

As an example, pattern $P = \{a\} \rightarrow \{b, c\} \rightarrow \{d\}$ is contained in sequence $s_1 = \langle sid1, \{e\}, \{a, d\}, \{g\}, \{b, c, f\}, \{d\}\rangle$ because $\{a\} \subseteq \{a, d\}$, $\{b, c\} \subseteq \{b, c, f\}$, and $\{d\} \subseteq \{d\}$. Hence, $P \sqsubseteq s_1$. On the other hand, $P \not\sqsubseteq s_2 = \langle sid2, \{a, d\}, \{d\}, \{b, c, f\}\rangle$ because $\{b, c\}$ occurs before $\{d\}$ in $P$, which is not the case in $s_2$.

A database $\mathcal{D}$ is composed of a number of sequences whose end states are known. We use $D_C$ to represent the set of sequences in $D$ that end with state $C$, and $D_{\overline{C}}$ to represent the set of sequences in $D$ that does not end with state $C$. Thus, $D = D_C \cup D_{\overline{C}}$. The sequences in $D_{\overline{C}}$ do not necessarily have the same end states, that is, there can be more than two end states in the database. We say that sequences with end state $C$ are in class $C$.

The support of a pattern $P$ in class $C$ (represented by $sup(P, C)$) is defined as the number of sequences in $D_C$ that match $P$. If $sup(P, C)$ is no less than a user specified support threshold $\rho_s$, we say $P$ is a frequent pattern in class $C$.

We use $sup(P, \overline{C})$ to represent the number of sequences in $D_{\overline{C}}$ that match $P$. The confidence of a pattern $P$ resulting in class $C$ is defined as

$$conf(P, C) = \frac{sup(P, C)}{sup(P, C) + sup(P, \overline{C})}.$$

Given a pattern $P = X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_n$, if $conf(X_1, C) \leq conf(X_1 \rightarrow X_2, C) \leq \ldots \leq conf(P, C)$, we say $P$ satisfies the progressive confident condition.

Finally, if a pattern $P$ satisfies the following four conditions

$$sup(P, C) \geq \rho_s; \quad (1)$$

$$conf(X_1, C) \geq min\_conf1; \quad (2)$$

$$conf(X_1, C) \leq conf(X_1 \rightarrow X_2, C) \leq \ldots \leq conf(X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_n, C) \quad (3)$$

$$conf(X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_n, C) \geq min\_conf2; \quad (4)$$

where $min\_conf1$, $min\_conf2$ and $\rho_s$ are three user specified parameters, we say $P$ together with its confidence values (i.e., $conf(X_1), conf(X_1 \rightarrow \ldots \rightarrow X_i, C)$, where $1 < i \leq n$) is a progressive confident rule.

## 2.2 Concise Set of PCR

We observe that the set of all progressive confident rules contains redundant information. In this section, we discuss how a concise set of progressive confident rules can be obtained.

**Example 1.** Suppose $P = \{a\} \rightarrow \{b, c\}$ and $Q = \{a\} \rightarrow \{b, c\} \rightarrow \{d\}$ are two progressive confident rules. If $conf(P, C) = 1$, by definition of confidence, $conf(Q, C)$ must also be 1, and the information captured in $Q$ is redundant.

**Example 2.** Suppose $P = \{a\} \rightarrow \{b, c\}$ and $Q = \{a\} \rightarrow \{b, c\} \rightarrow \{d\}$ are two progressive confident rules. If $conf(P, C) < 1$, then all information in $P$ is already contained in $Q$.

Given a pattern $Q$, we use $Q[i]$ to represent the $i$-th state (itemset) in $Q$, and $Q[i, j]$ ($j > i$) to represent the pattern $Q[i] \rightarrow Q[i+1] \rightarrow \ldots \rightarrow Q[j]$. For example, if $Q = \{a\} \rightarrow \{b, c\} \rightarrow \{d\}$, then $Q[1] = \{a\}$, $Q[2] = \{b, c\}$, $Q[3] = \{d\}$, and $Q[1, 2] = \{a\} \rightarrow \{b, c\}$.

A *base-pattern* of a pattern $Q$ is a pattern obtained by removing the last several states (itemsets) from $Q$. For example, if $Q = \{a\} \rightarrow \{b, c\} \rightarrow \{d\}$, then $\{a\} \rightarrow \{b, c\}$ is $Q$'s base-pattern, and $\{a\} \rightarrow \{b\}$ is not. If $P$ is a base-pattern of $Q$, we also say $Q$ is an *extender* of $P$.

Give a pattern $Q$, if $conf(Q, C) = 1$ and none of its base-pattern $P$ satisfies the condition $conf(P, C) = 1$, we say $Q$ is a *terminator pattern*.

Class $C$

| Sequence id | Observation | |
|---|---|---|
| | tid | Itemset |
| 1 | 1 | $\{a, e\}$ |
| | 2 | $\{b, c\}$ |
| | 3 | $\{d\}$ |
| 2 | 4 | $\{a\}$ |
| | 5 | $\{b, c\}$ |
| | 6 | $\{d, f\}$ |
| 3 | 7 | $\{a\}$ |
| | 8 | $\{g\}$ |
| | 9 | $\{e\}$ |

Classes not of $C$

| Sequence id | Observation | |
|---|---|---|
| | tid | Itemset |
| 4 | 11 | $\{a\}$ |
| | 12 | $\{b, c, d\}$ |
| | 13 | $\{e\}$ |
| 5 | 14 | $\{a\}$ |
| | 15 | $\{b, d\}$ |
| | 16 | $\{f\}$ |
| 6 | 17 | $\{a\}$ |
| | 18 | $\{d\}$ |
| | 19 | $\{b\}$ |

**Table 2: A sample database**

Given a set of patterns $V$ and a pattern $P \in V$, if $P$ is not a base-pattern of any other pattern $P'$ in $V$, we say $P$ is a *maximal pattern* in $V$.

In Example 1, all the extenders of a terminator pattern can be ignored without loss of information. Further, if a pattern is neither a terminator pattern nor a maximal pattern (such as $P$ in Example 2), then that pattern can be excluded from the concise set.

Based on the above discussion, we can obtain a concise set of rules from a set of progressive confident rules $R$ in two steps:

- Step 1. Delete all extenders of terminator patterns in $R$;
- Step 2. Remove non-maximal patterns.

We define the problem of mining progressive confident rules as finding the concise set of all progressive confident rules in a given database $\mathcal{D}$ and a class $C$.

# 3. MINING CONCISE SET OF PCR

One common issue for data mining problems is that the search space is huge. As a result, many mining algorithms make use of the *apriori* property to reduce the search space to a manageable one. The apriori property states that if a pattern $P$ satisfies the mining requirement, then all its subpatterns also meet the requirement. For example, the `Apriori` algorithm for finding large itemsets [14] and the `GSP` algorithm for mining sequential patterns [15] are both based on this property.

Unfortunately, the apriori property does not hold in the progressive confident condition (i.e. $conf(P[1], C) \leq conf(P[1, 2], C) \leq \ldots \leq conf(P, C)$). We illustrate this with a simple example.

Consider the database in Table 2. Let the three user input parameters be set as $\rho_s = 2$, $min\_conf1 = 0.5$ and $min\_conf2 = 0.9$.

The pattern $P = \{a\} \rightarrow \{b, c\} \rightarrow \{d\}$ is a progressive confident rule because

- $sup(P, C) = 2 \geq \rho_s$;
- $conf(\{a\}, C) = \frac{3}{3+3} = 0.5 \geq min\_conf1$;
- $conf(\{a\} \rightarrow \{b, c\}, C) = \frac{2}{2+1} = 0.67 > conf(\{a\}, C)$; $conf(\{a\} \rightarrow \{b, c\} \rightarrow \{d\}, C) = \frac{2}{2+0} = 1 \geq conf(\{a\} \rightarrow \{b, c\}, C)$;
- $conf(\{a\} \rightarrow \{b, c\} \rightarrow \{d\}, C) = 1 \geq min\_conf2$.

However, $P$'s subpattern $Q = \{a\} \rightarrow \{b\}$ does not satisfy the progressive confident condition, because $conf(\{a\} \rightarrow \{b\}, C) = \frac{2}{2+3} = 0.4 < conf(\{a\}, C) = 0.5$. Therefore, the apriori property does not hold.

Another property that can be used to prune the search space is the prefix anti-monotonic property [12]. This property states that if a pattern $P_1$ meets the mining requirement and another pattern $P_2$ is obtained by removing some ending items from $P_1$, then $P_2$ also satisfies the mining requirement. In our example, $Q$ is obtained by removing ending items $c$ and $d$ from $P$. Although $P$ is a progressive confident rule, $Q$ does not satisfy the progressive confident condition. Hence, the prefix anti-monotonic property is also not applicable for our problem here.

We put forward two theorems to help reduce the huge search space.

THEOREM 1. *Suppose a pattern $P$ satisfies the following three conditions.*

$$\begin{cases} sup(P, C) \geq \rho_s; \\ conf(P[1], C) \geq min\_conf1; \\ conf(P[1], C) \leq conf(P[1, 2], C) \leq \ldots \leq conf(P, C). \end{cases}$$

*If $Q$ is a base-pattern of $P$, then $Q$ also satisfies the three conditions.*

$$\begin{cases} sup(Q, C) \geq \rho_s; \\ conf(Q[1], C) \geq min\_conf1; \\ conf(Q[1], C) \leq conf(Q[1, 2], C) \leq \ldots \leq conf(Q, C). \end{cases}$$

**Proof:** Let the base-pattern $Q$ be $Q = P[1, i](1 \leq i < |P|)$.

Because $Q$ is a base-pattern of $P$, we have

$$sup(Q, C) \geq sup(P, C) \geq \rho_s,$$

$Q$ meets the first condition.

From $Q = P[1, i]$, we get

$$conf(Q[1], C) = conf(P[1], C) \geq min\_conf1.$$

$Q$ also satisfies the second requirement.

Since $Q = P[1, i]$, we have

$$\begin{aligned} & conf(Q[1], C) = conf(P[1], C) \\ \leq \; & conf(P[1, 2], C) = conf(Q[1, 2], C) \\ \leq \; & \ldots \\ \leq \; & conf(P[1, i], C) = conf(Q, C). \end{aligned}$$

So $Q$ also obeys the progressive confident condition.

Hence, Theorem 1 is true. □

Theorem 1 tells us that if a pattern $Q$ does not satisfy either one of the support requirement, $min\_conf1$ requirement or progressive confident condition, nor is any of $Q$'s extenders. Thus, if we find such a pattern $Q$ in the mining process, we do not need to consider $Q$'s extenders as potential progressive confident rules. This will prune off a huge amount of search space.

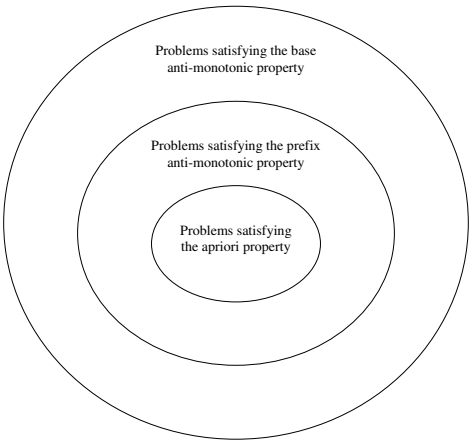


**Figure 1: Relationships of three properties**

We denote the property described in Theorem 1 as the *base anti-monotonic property*. All the mining problems that satisfy the apriori property also satisfy the prefix anti-monotonic property. All mining problems that obey the prefix anti-monotonic property also obey the base anti-monotonic property. Figure 1 shows the relationship among these three properties. Therefore, an algorithm that solves a problem with the apriori property is not necessarily applicable to the problems with the prefix anti-monotonic property or the base anti-monotonic property. Similarly, an algorithm that solves a problem with the prefix anti-monotonic property may not be suitable for problems with the base anti-monotonic property.

In [12], the authors propose a projection framework for mining sequential patterns with constraints that satisfy the prefix anti-monotonic property. The efficiency of the framework lies in the prefix anti-monotonic property. When this property holds, by continuous database projections on individual items, no subset testing and candidate generation are required. Further, the number of projections in each level is bounded by the number of frequent items in the database, i.e., O(N).

Our mining problem does not follow the prefix anti-monotonic property. With Theorem 1, we can modify the framework in [12] by projecting on itemsets, not on items. However, this will be inefficient. When making projections on itemsets, subset testing will be unavoidable. Candidate generation is necessary, otherwise one needs to enumerate all the itemsets in a database sequence. Further, the number of projections for each level is bounded by the number of frequent itemsets in the database, which is O($2^n$), much larger than O(n) when projecting on items. Therefore, the projection framework in [12] is not suitable here.

If a pattern $P$ satisfies the progressive confident condition and the support requirement, we need to consider $P$'s extender $P \rightarrow X$. At this point, $conf(P, C)$ is already known. In order to check whether $conf(P \rightarrow X, C) \geq conf(P, C)$, we should know $conf(P \rightarrow X, C)$. According to definition of $conf(P \rightarrow X, C)$, one need to calculate both $sup(P \rightarrow X, C)$ and $sup(P \rightarrow X, \overline{C})$. The following theorem helps us to avoid computing $sup(P \rightarrow X, C)$ when $P \rightarrow X$ does not satisfies the progressive confident condition,

THEOREM 2. *Given a pattern $P$ and a state $X$, if*

$conf(P,C) < 1$ *and* $conf(P \rightarrow X, C) \geq conf(P,C)$*, we have*

$$sup(P,C) \geq sup(P \rightarrow X, \overline{C}) \frac{conf(P,C)}{1 - conf(P,C)},$$

*and*

$$sup(X,C) \geq sup(P \rightarrow X, \overline{C}) \frac{conf(P,C)}{1 - conf(P,C)}.$$

**Proof:** From $conf(P \rightarrow X, C) \geq conf(P,C)$, we get

$$1 - conf(P \rightarrow X, C) \leq 1 - conf(P,C).$$

Therefore,

$$\frac{conf(P \rightarrow X, C)}{1 - conf(P \rightarrow X, C)} \geq \frac{conf(P,C)}{1 - conf(P,C)}.$$

By definition,

$$\frac{conf(P \rightarrow X, C)}{1 - conf(P \rightarrow X, C)} = \frac{sup(P \rightarrow X, C)}{sup(P \rightarrow X, \overline{C})}.$$

So

$$\frac{sup(P \rightarrow X, C)}{sup(P \rightarrow X, \overline{C})} \geq \frac{conf(P,C)}{1 - conf(P,C)}.$$

Or

$$sup(P \rightarrow X, C) \geq sup(P \rightarrow X, \overline{C}) \frac{conf(P,C)}{1 - conf(P,C)}$$

.

Therefore,

$$\begin{aligned} sup(P,C) &\geq sup(P \rightarrow X, C) \\ &\geq sup(P \rightarrow X, \overline{C}) \frac{conf(P,C)}{1 - conf(P,C)}, \end{aligned}$$

and

$$\begin{aligned} sup(X,C) &\geq sup(P \rightarrow X, C) \\ &\geq sup(P \rightarrow X, \overline{C}) \frac{conf(P,C)}{1 - conf(P,C)}. \end{aligned}$$

Hence, Theorem 2 is true. □

Theorem 2 provides a pruning test on whether to extend a pattern $P$ to $P \rightarrow X$. If $sup(P,C)$ or $sup(X,C)$ is not large enough (compared with $sup(P \rightarrow X, \overline{C}) \frac{conf(P,C)}{1-conf(P,C)}$), we have $conf(P \rightarrow X, C) < conf(P,C)$. This is a contradiction to the progressive confident condition, and we do not need to extend $P$ to $P \rightarrow X$, which saves the time for counting $sup(P \rightarrow X, C)$.

Theorem 2 does not handle the case when $conf(P,C) = 1$. This is inconsequential since $conf(P,C) = 1$ implies that $P$ is a terminator pattern. No extenders of $P$ will be in the concise set, and we do not need to consider $P \rightarrow X$ in the algorithm.

## 3.1 Algorithm

Based on Theorem 1 and Theorem 2, we design an efficient depth-first algorithm for mining the concise set of all progressive confident rules. We call it FMP (Fast Mining of Progressive confident rules).

Figure 2 gives the Algorithm FMP. The algorithm first finds all frequent itemsets in class $C$. Then for each frequent itemset that satisfies $min\_conf1$ requirement, FMP processes its extenders in a depth-first order to search for other potential progressive confident rules.

```
1   Algorithm FMP (D, ρ_s, min_conf1, min_conf2)
2     Find frequent itemsets in class C:
      L_X = {X | sup(X,C) ≥ ρ_s}
3     ∀X ∈ L_X
4       calculate conf(X,C)
5       if conf(X,C) ≥ min_conf1
6         extend(X);

7   Function extend (pattern P)
8     if conf(P,C) = 1
9       output P;
10      return;
11    ext=false;
12    ∀X ∈ L_X
13      count sup(P → X, C̄)
14      stand = sup(P → X, C̄) conf(P,C)/(1−conf(P,C))
15      if sup(P,C) < stand
16        continue with next itemset in L_X;
17      if sup(X,C) < stand
18        continue with next itemset in L_X;
19      count sup(P → X, C)
20      if sup(P → X, C) < ρ_s
21        continue with next itemset in L_X;
22      count conf(P → X, C)
23      if conf(P → X, C) < conf(P,C)
24        continue with next itemset in L_X;
25      ext=true;
26      extend(P → X);
27    if (ext = false ∧ conf(P) ≥ min_conf2)
28      output P;
```

**Figure 2: Algorithm FMP**

The core of FMP lies in the $extend$ function (lines 7–28 of Figure 2). Given a pattern $P$, the function finds out all extenders of $P$ that are in the concise set of progressive confident rules.

In the $extend$ function, FMP checks whether $conf(P,C)$ is equal to 1. If so, $P$ is a terminator pattern and we can ignore all $P$'s extenders because they are not in the concise set. $P$ will be output. If $conf(P,C) \neq 1$, we check whether we need to append a state $X$ to the end of $P$ with the help of Theorem 2 (lines 13–18). If $X$ passes the test, we check whether $P \rightarrow X$ meets the support requirement and progressive confident condition (lines 19–24). If yes, we recall the $extend$ function with new parameter value $P \rightarrow X$. Finally, if $P$ could not be extended anymore (i.e. $P$ is a maximal pattern) and $conf(P,C) \geq min\_conf2$, we output $P$ and the function terminates.

In the $extend$ function, the support of $P \rightarrow X$ is counted from the ID lists of $P$ and $X$, not by scanning the database. The ID list of a pattern records the sequences and the time stamps in which a pattern appears. The concept is also used by the SPADE algorithm [17] for mining frequent sequences.

The depth-first order and the careful control of the $extend$ function guarantees that the rules in the output belongs to the concise set.

The efficiency of the FMP algorithm mainly comes from two aspects. One is the utilization of the two theorems to effectively reduce the search space. The other is by incorporating the concise set analysis in the mining process to avoid checking the extenders of terminator patterns, which further reduces the search space.

| Parameter | Description | Value |
|---|---|---|
| $\|D\|$ | Number of sequences in database $\mathcal{D}$ | 100,000 |
| $\|D_C\|$ | Number of sequences in class $C$ | 50,000 |
| $\|D_{\overline{C}}\|$ | Number of sequences not in class $C$ | 50,000 |
| $O$ | Average number of observations per sequence | 8 |
| $T$ | Average number of items per observation | 2.5 |
| $S$ | Average number of itemsets in maximal potentially frequent patterns | 6 |
| $I$ | Average size of itemsets in maximal potentially frequent patterns | 1.25 |
| $N_S$ | Number of maximal potentially frequent patterns | 500 |
| $N_I$ | Number of maximal potentially frequent itemsets | 2,500 |
| $N$ | Number of items | 1,000 |

**Table 3: Parameters and values for data generation**

# 4. EXPERIMENTS

We carried out experiments to study the performance of Algorithm FMP and evaluate the effectiveness of progressive confident rules in classification.

We compare Algorithm FMP with a naive algorithm DMP (Direct Mining of Progressive confident rules) shown in Figure 3. Algorithm DMP first makes use of the SPADE algorithm to find out all frequent patterns (line 2). It then removes those frequent patterns that do not satisfy the other requirements of progressive confident rules (lines 3–10). Finally, DMP computes the concise set of rules and output it (lines 11–12). Note that both FMP and DMP utilize the ID lists to count the support of a pattern, which makes their performance comparison reasonable.

We implemented both algorithms in C++. The experiment is done on a Sun Fire 4800 midframe sever with 750MHz Ultra Sparc III CPU and 1GB RAM, running SunOS 5.8.

```
1   Algorithm DMP (D, ρ_s, min_conf1, min_conf2)
2     find L = {P | sup(P, C) ≥ ρ_s}
3     ∀P ∈ L
4       if conf(P[1], C) < min_conf1
5         delete P
6       else if conf(P, C) < min_conf2
7         delete P
8       else for i = 1 to |P| − 1
9         if conf(P[1, i], C) > conf(P[1, i + 1], C)
10          delete P
11    calculate concise set of L
12    output the result
```

**Figure 3: Algorithm DMP**

## 4.1 Synthetic Data Generation

We generate the test data in the following three steps.

In the first step, we generate a table of potentially frequent itemsets. The size of each itemset is computed as a Poisson distributed random number with mean equal to $I$. Except for the first itemset, every other itemset shares some common items with its immediate preceding one. The number of items to be shared is controlled by an exponentially distributed random number. In this step, we generate $N_I$ itemsets from $N$ different items.

Each generated itemset is assigned a confidence value $c$. This confidence value determines the probability an itemset will appear in class $C$. $c$ is calculated by the following formula:

$$c = \begin{cases} r & \text{if } 0 \le r \le 1; \\ 0 & \text{if } r < 0; \\ 1 & \text{if } r > 1, \end{cases}$$

where $r$ is a normal-distributed random number with mean equal to 0.5.

In the second step, we generate two pattern tables for dataset $D_C$ (corresponding to class $C$) and $D_{\overline{C}}$ (corresponding to classes other than $C$), respectively. We call the two pattern tables $T_C$ and $T_{\overline{C}}$. The length of a pattern $p$ is determined by a Poisson distributed random number with mean equal to $S$. The content of $p$ is a series of itemsets generated in the first step. The itemsets are selected randomly from the itemset table. If the confidence of a selected itemset is $c$, we append it to a pattern belonging to $T_C$ with a probability of $c$, and append it to a pattern in $T_{\overline{C}}$ with the probability of $1 - c$. When a pattern reaches its length, we insert it into the corresponding pattern table, and continue to generate the next pattern for the same pattern table. In this step, we generate $N_s$ patterns for both $T_C$ and $T_{\overline{C}}$.

For each generated pattern $p$, we assign a weight and a corruption level to it. The weight represents the probability of $p$ being picked for generating a sequence in the third step, and the corruption level controls how many items will be randomly dropped from $p$ before it is inserted into a sequence.

In the third step, we generate sequences in our test database. Sequences in $D_C$ is computed from pattern table $T_C$, and sequences in $D_{\overline{C}}$ is generated from $T_{\overline{C}}$. When generating a sequence, we first calculate its length, which is determined by a Poisson distributed variable whose mean is $O$, and the average number of items in its itemsets, which is a Poisson distributed variable with mean equal to $T$. We then randomly choose a pattern from the corresponding pattern table to fill in it. If the pattern is too large for the sequence, we discard it in half the cases, and push it into the sequence anyway in the rest cases. In this step, we generate $|D_C|$ and $|D_{\overline{C}}|$ sequences from Tables $T_C$ and $T_{\overline{C}}$ respectively.

Table 3 summarizes the parameters of the generator. It also shows the default value used in our experiments.

$|D| = 100,000$, $min\_conf1 = 0.5$, $min\_conf2 = 0.9$

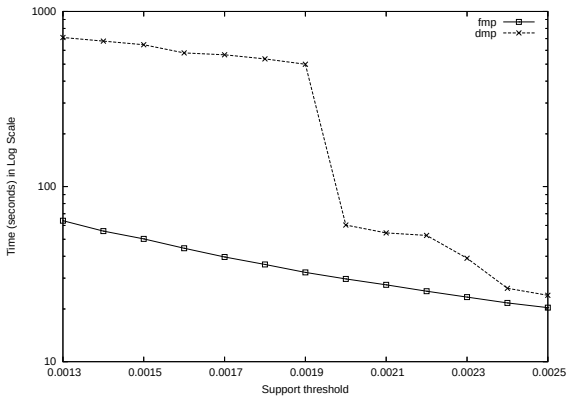


**Figure 4: Performance of DMP and FMP vs. support threshold**

## 4.2 Effect of Varying Support Threshold

In the first experiment, we test the performance of the two algo-

rithms under different support thresholds. The result is presented in Figure 4.

The x-axis of the figures is the support threshold, and the y-axis is the running time in log scale. We see that FMP runs much faster than DMP. When support threshold is low ($\rho_s \leq 0.0019$), FMP is one order faster than DMP. This is because with a low $\rho_s$, there are many frequent patterns, and DMP has to mine all of them. While FMP only needs to deal with part of them that satisfy not only support requirement but also confidence requirements.

When the support threshold increases, there are fewer frequent patterns, and DMP requires less effort to mine them, thus narrowing the gap between the two algorithms. However, FMP is still about 2 times faster than DMP.

## 4.3 Effect of Varying Confidence Thresholds

Next, we examine the influence of the parameters $min\_conf1$ and $min\_conf2$ on the two algorithms.

Figure 5 shows the running time of DMP and FMP in log scale when $min\_conf1$ changes from 0.4 to 0.65. We see that when $min\_conf1$ increases, the running times of both algorithms decrease. For DMP, with a higher $min\_conf1$, more patterns could be pruned before resorting them to progressive confident condition test and concise set computation. Thus, the running time decreases. When $min\_conf1$ increases from 0.4 to 0.65, DMP saves $(566.36 - 481.23)/566.36 = 15\%$ execution time.

For FMP, it incorporates the test of $min\_conf1$ into the mining process, i.e., it does not extend an itemset if its confidence is less than $min\_conf1$. Hence, the influence of $min\_conf1$ on FMP is higher than that on DMP. When $min\_conf1$ changes from 0.4 to 0.65, FMP reduces $(42.63 - 29.40)/42.63 = 31\%$ running time, larger than $15\%$ saved by DMP.

$|D| = 100,000$, $min\_conf2 = 0.9$, $\rho_s = 0.0018$

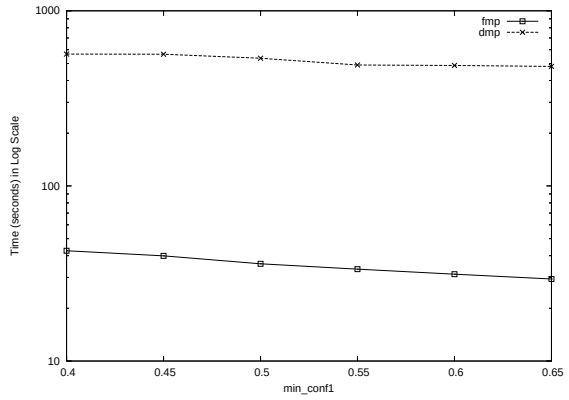


**Figure 5: Performance of DMP and FMP vs. $min\_conf1$**

Figure 6 shows the execution time of DMP and FMP when $min\_conf2$ changes from 0.75 to 1. The performance of both two algorithms stays relatively steady. Since FMP does not use $min\_conf2$ for pruning, $min\_conf2$ has nearly no effect on its performance. For DMP, the savings on progressive confident condition check and concise set calculation by deleting patterns which do not meet the $min\_conf2$ requirement is also negligible.

## 4.4 Effect of Parameters $O$, $T$, $S$ and $I$

We also study the influence of generator parameters $O$, $T$, $S$ and $I$ on the performance of the algorithms. We first fix the values of $T$, $S$, $I$, and change $O$ (average number of observations per sequences) from 6 to 12. The execution time of DMP and FMP is shown in Figure 7. We see that as $O$ increases, the running time of both algorithms increases as well. Recall that $O$ represents the average length of sequences. A larger $O$ implies more frequent patterns and more progressive confident rules, and hence additional execution time. The increase in running time of FMP is much smaller than that of DMP. The reason is that FMP could efficiently avoid processing many new frequent patterns (caused by a larger $O$) that are not new progressive confident rules in the concise set; while DMP has to work on all new frequent patterns.

Next, we fix the values of $O$, $S$, $I$, and change $T$ (the average number of items in the itemset). Figure 8 shows the performance of DMP and FMP when $T$ changes from 2 to 5. We see that the running time of DMP varies greatly under different $T$ values. Recall that when generating a sequence in the synthetic database, we repeatedly append a random pattern to the end of the sequence until the sequence exceeds its size limit. At this point, the current pattern will be inserted into the sequence or be discarded randomly. Since $T$ is used to control the size limit of a sequence, different $T$ values result in different databases and different set of frequent sequences. So the performance of DMP, which mines all frequent sequences, varies as $T$ changes. On the other hand, $T$ does not influence the

$|D| = 100,000$, $min\_conf1 = 0.5$, $\rho_s = 0.0018$

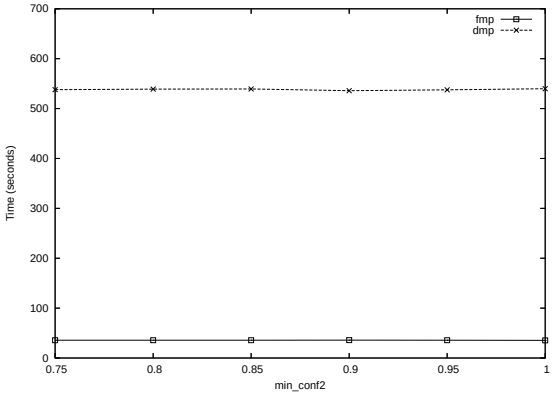


**Figure 6: Performance of DMP and FMP vs. $min\_conf2$**

$|D| = 100,000$, $min\_conf1 = 0.5$, $min\_conf2 = 0.9$
$T = 2.5$, $S = 6$, $I = 1.25$, $\rho_s = 0.002$

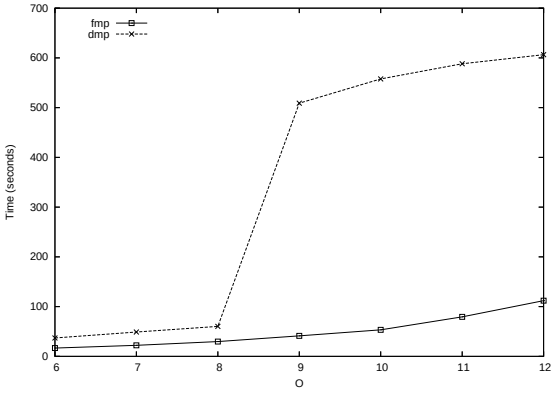


**Figure 7: Performance of DMP and FMP vs. $O$**

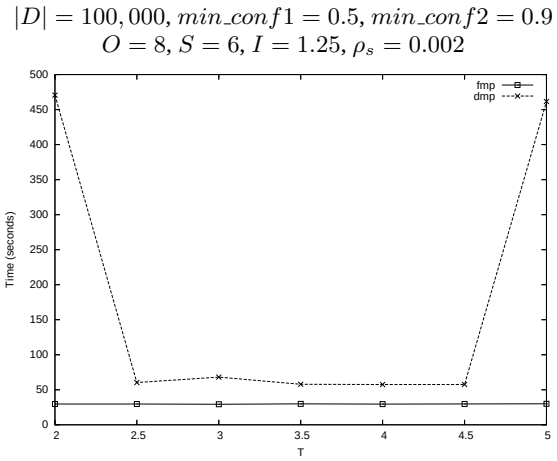


**Figure 8: Performance of DMP and FMP vs. $T$**

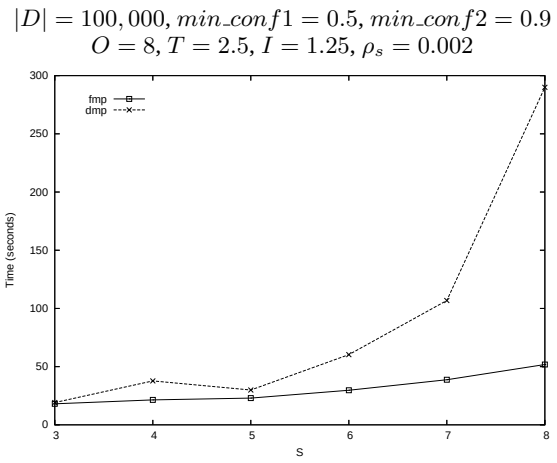


**Figure 9: Performance of DMP and FMP vs. $S$**

pattern table, which contains potential progressive confident rules. By mining progressive confident rules directly, the performance of FMP is not influenced as much as DMP is.

In the next experiment, we fix $O$, $T$, $I$, and change $S$ (average length of potentially maximal frequent patterns) from 3 to 8. The experiment result is shown in Figure 9. Since a larger $S$ implies more frequent patterns and more progressive confident rules, both algorithms need more processing time. Similar to Figure 7, as $S$ increases, the running time of FMP increases slowly, while that of DMP increases dramatically.

We also study the performance of DMP and FMP when the average number of items in an itemset ($I$) changes. With a larger $I$, the frequencies of items increase in the database, which results in more frequent patterns and more progressive confident rules. Hence, the execution times of both DMP and FMP increase as $I$ becomes larger (see Figure 10). Again, the increasing rate of FMP is slower than that of DMP, because FMP incorporates other condition checks in the mining process to prune the search space.

## 4.5 Scalability

Finally, we test the scalability of DMP and FMP with the number of sequences ranging from 100,000 to 1,000,000. The result is

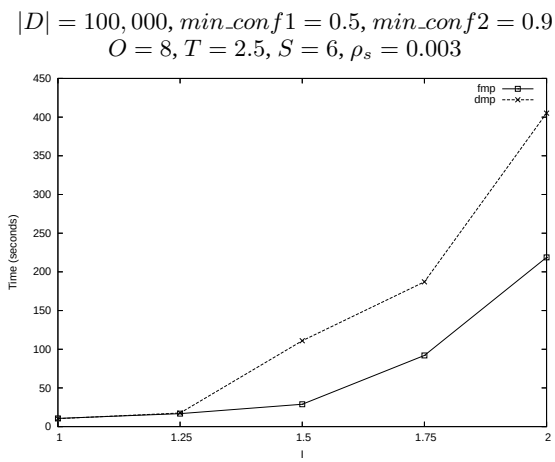


**Figure 10: Performance of DMP and FMP vs. $I$**

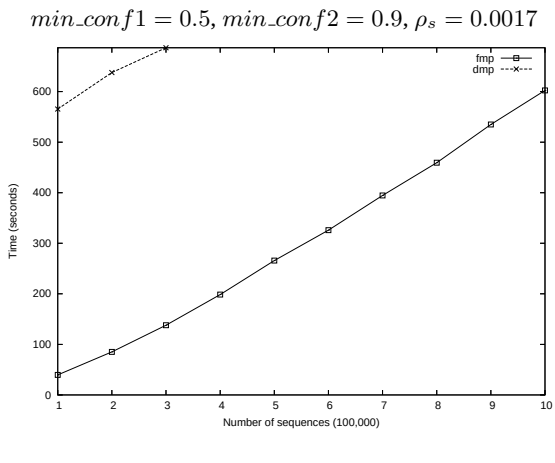


**Figure 11: Performance of DMP and FMP vs. No. of sequences**

shown in Figure 11.

We see that FMP scales linearly with the number of sequences. For DMP, it scales linearly at first. However, when the number of sequences reaches 400,000, it runs out of memory. The reason is that the increase in the number of sequences results in longer ID lists of patterns. DMP, which mines all frequent patterns before the filtering part, requires a lot of memory to store the long ID lists. While FMP, which combines the mining part and the filtering part, is able to prune a lot of patterns and their ID lists directly.

# 5. APPLICATION OF PCR IN CLASSIFICATION

One application of the progressive confident rule is to predict a future state of an object based on its past state sequence. In this section, we describe how progressive confident rules can be utilized in three representative classifiers C4.5 [13], Support Vector Machine (SVM) [4] and Bayes Classifier (BC) [6].

We download C4.5, SVM and BC from websites [3, 16, 2], respectively. All three classifiers C4.5, SVM and BC require input objects being described by {feature, value} pairs. For objects with state sequences, the classifiers can run directly on their last states. However, in order to utilize the state sequences in the classifiers,

| **data** | class $C$ | class $\overline{C}$ | **total** |
|---|---|---|---|
| training cases | 25,000 | 25,000 | 50,000 |
| test cases | 25,000 | 25,000 | 50,000 |
| **classifier accuracy** | class $C$ | class $\overline{C}$ | **avg.** |
| C4.5 | 74.08% | 79.44% | 76.76% |
| C4.5_PCR | 77.70% | 99.62% | 88.66% |
| SVM | 75.99% | 77.49% | 76.74% |
| SVM_PCR | 99.63% | 77.69% | 88.66% |
| BC | 88.06% | 62.23% | 75.15% |
| BC_PCR | 99.12% | 74.60% | 86.86% |

**Table 4: Classification result on synthetic data**

some mappings have to be done first.

We map each progressive confident rule $r$ in the concise set to a new feature $f_r$. If the state sequence of an object matches $r$, the value of $f_r$ is set to 1. Otherwise, it is set to 0. Given $n$ progressive confident rules, the state sequence of an object is transformed to $n$ {feature, value} pairs. The three classifiers are run on the converted new feature set.

Given a training dataset and a test dataset of state sequences, we first mine the concise set of progressive confident rules in the training data for every class (end state). We then utilize the discovered rules to convert the original training data and test data to {feature, value} pair databases. Finally, we run the classifiers on the transformed datasets.

We carry out experiments on both synthetic and real data to test the performance of classifiers incorporated with progressive confident rules (PCRs). We use C4.5_PCR (C4.5 with Progressive Confident Rules), SVM_PCR and BC_PCR to represent the three classifiers incorporated with PCRs, respectively. C4.5, SVM, and BC is used to represent the three classifiers running on the last states of objects, i.e. not incorporating PCRs.

The first set of experiment is carried out on the synthetic dataset generated in Section 4.1. There are two classes $C$ and $\overline{C}$ in this dataset. Each class has 50,000 cases. We select 50% cases from each class for training, and use the remaining cases for testing. The mining algorithm generated 1,154 progressive confident rules from two classes.

Table 4 shows the classification accuracy of C4.5, SVM, BC, C4.5_PCR, SVM_PCR, and BC_PCR in two classes and in the whole test data. We see that classifiers incorporated with PCRs, i.e. C4.5_PCR, SVM_PCR and BC_PCR, outperform their counterparts C4.5, SVM, BC. On average, they could achieve about 12% more accuracy.

We also carried out experiments on a real-life dataset which captures the retinal examination data of 10,845 diabetic patients. Each patient has around 2-6 examination records at different times. All records for a patient forms his state sequence. Items of this dataset are related to symptoms such as drusen, cataract, and hypertension. We are interested in whether a patient has maculopathy in his last exam record (state). There are two classes in this dataset. In one class, the patients' last examination records show maculopathy. We call it class `FMAC-Y`. There are 111 patients in this class. In the other class, maculopathy does not appear in the last exams. We call it `FMAC-N`. It contains 10,734 patients. Suppose a patient has 5 exam records. We predict whether maculopathy appears in the 5th record from the first 4 records. We say a prediction is correct if it is the same as what is shown in the last exam record. Otherwise, the prediction is wrong.

| **data** | FMAC-N | FMAC-Y | **total** |
|---|---|---|---|
| training cases | 100 | 100 | 200 |
| test cases | 10634 | 11 | 10645 |
| **classifier accuracy** | FMAC-N | FMAC-Y | **g_mean** |
| C4.5 | 57.22% | 9.09% | 22.81% |
| C4.5_PCR | 76.47% | 27.27% | 45.67% |
| SVM | 67.84% | 63.64% | 65.70% |
| SVM_PCR | 64.25% | 90.91% | 76.42% |
| BC | 89.21% | 18.18% | 40.27% |
| BC_PCR | 60.06% | 90.91% | 73.89% |

**Table 5: Classification result on diabetic retinal data**

This real-life dataset is very biased. The size of class `FMAC-N` is about 100 times that of class `FMAC-Y`. If a classifier assigns all test cases to `FMAC-N`, the average accuracy is high. However, no cases in `FMAC-Y` could be recognized. For such biased data, people usually use *geometric mean ($g\_mean$)* to measure the total classification accuracy for all classes.

$$g\_mean = \sqrt{TP \times TN},$$

where $TP$ is recall or true positive rate, and $TN$ is true negative rate. When a classifier gives all test cases the same label, either $TP$ or $TN$ is 0, so $g\_mean$ is 0.

We set the training size of two classes be 90% that of class `FMAC-Y`, i.e. 100. There are 1,268 progressive confident rules in the training data. Table 5 shows the classification accuracy of the classifiers in two classes and the $g\_mean$ values. From $g\_mean$ value, we see that C4.5 and BC perform poorly. While C4.5_PCR and BC_PCR could improve the classification accuracy of their counterparts greatly. SVM_PCR also outperforms SVM with a wide margin.

In the above experiments, all progressive confident rules in the concise set are utilized in the classification. We also run an experiment on the diabetic retinal data using only part of the rules. For rules in each class, we select the rules with top K support values. So the total number of rules being used is $2K$. Figure 12 shows the $g\_mean$ values of C4.5_PCR, SVM_PCR and BC_PCR under different $K$ values. When $K = 650$, it is the case when all rules are utilized. We see that the accuracy of SVM_PCR and BC_PCR stays relatively steady. But the performance of C4.5_PCR changes as $K$ varies. The $g\_mean$ values of C4.5, SVM and BC are not related to $K$. We see that under all $K$ values, C4.5_PCR, SVM_PCR and BC_PCR have a better accuracy than their counterparts C4.5, SVM and BC, respectively.

# 6. RELATED WORK

Our work is related to sequential pattern mining problem which is first proposed by Agrawal et al. in 1995 [1]. The target is to mine frequent sequences from a transactional database. Efficient algorithms for this problem include `GSP` [15], `SPADE` [17] and `PrefixSpan` [11]. `GSP` utilizes the apriori property to prune the huge search space [15]. It can also be used to solve other generalized versions of the frequent-sequence mining problem. For example, a user can specify a sliding time window. Items that occur in itemsets that are within the sliding time window could be considered as to occur in the same itemset. `SPADE` [17] algorithm works on a vertical representation of the database. In the vertical representation, every item in the database is associated with an id-list. Like the inverted-list, the id-list of an item stores where the item appears

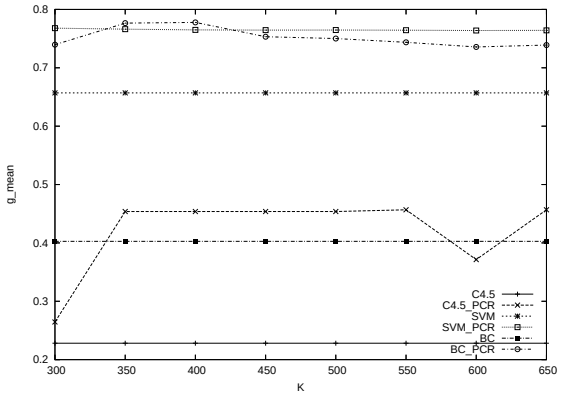


**Figure 12: Classification result on diabetic retinal data, selected progressive confident rules**

in the database. A vertical database is composed of the id-lists of all items. SPADE achieves efficiency by computing the supports of sequences from id-lists directly. PrefixSpan [11] mines frequent sequences by intermediate database generation. It applies a projection framework. By continuously projecting databases on individual items, time consuming subset testing and candidate generating could be avoided. Our work is different from theirs in that we mine frequent sequences with extra constraints.

In [7], Garofalakis et al. propose the use of regular expressions as a tool for the end-users to specify the kind of frequent sequences the system should return. A family of algorithms are proposed to mine those frequent sequences with regular expression constraints. Jian et al. [12] examine mining sequential patterns with constraints that obey the prefix anti-monotonic property. They propose the prefix-growth algorithm to solve the problem. Like PrefixSpan, prefix-growth also utilizes a database projection method. Our work is different from theirs in that our constraints neither follow regular expressions nor obey the prefix anti-monotonic property.

Liu et al. propose to integrate classification and association rule mining [10]. A new classifier called CBA is built on top of the discovered association rules in the database. In [8], the authors study the sequence classification problem. They first mine frequent sequences with restrictions such as longest length and maximal itemset size. They then use them in the classification. Emerging patterns [5] are itemsets whose supports in one class is significantly larger than those in other classes. In [9], Li et al. make use of emerging patterns for classification. Our paper proposes a new kind of pattern called progressive confident rules, which could also be utilized for classification.

# 7. CONCLUSION

In this paper, we have described a new kind of pattern called progressive confident rules. The rules capture the state change of objects that leads to a certain end state with increasing confidence. We have given a formal definition of progressive confident rules and their concise set. We put forward new pruning strategies to reduce the huge search space. A novel depth-first mining algorithm FMP is designed based on the new pruning strategies. FMP incorporates the concise set analysis in the mining process to further prune the search space. Experiment result shows that FMP is one order faster than a straight forward method DMP.

We have also illustrated an application of progressive confident rules in predicting a future state of an object given its past state sequence. We describe how to utilize the progressive confident rules in three representative classifiers C4.5, SVM and BC. Experiment result on both synthetic and real data shows that all three classifiers incorporated with progressive confident rules could improve the classification accuracy greatly.

# 8. REFERENCES


[1] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proc. of the 11th Int'l Conference on Data Engineering*, Taipei, Taiwan, March 1995.

[2] http://fuzzy.cs.uni-magdeburg.de/ borgelt/bayes.html.

[3] http://www.rulequest.com/.

[4] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[5] Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: discovering trends and differences. In *Proc. of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'99)*, 1999.

[6] Nir Friedman and Moises Goldszmidt. Building classifiers using bayesian networks. In *AAAI/IAAI, Vol. 2*, pages 1277–1284, 1996.

[7] Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *Proceedings of the 25th International Conference on Very Large Data Bases*, Edinburgh, Scotland, UK, September 1999.

[8] Neal Lesh, Mohammed J. Zaki, and Mitsunori Ogihara. Mining features for sequence classification. In *Proc. of KDD-99*, 1999.

[9] Jinyan Li, Guozhu Dong, Kotagiri Ramamohanarao, and Limsoon Wong. DeEPs: A new instance-based lazy discovery and classification system. *Machine Learning*, 54, 2004.

[10] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proc. of KDD-98*, 1998.

[11] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proc. 17th IEEE International Conference on Data Engineering (ICDE)*, Heidelberg, Germany, April 2001.

[12] Jian Pei, Jiawei Han, and Wei Wang. Mining sequential patterns with constraints in large databases. In *Proc. of CIKM'02*, 2002.

[13] John Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., 1993.

[14] T. Imielinski R. Agrawal and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD International Conference on Management of Data*, page 207, Washington, D.C., May 1993.

[15] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the 5th Conference on Extending Database Technology (EDBT)*, Avignion, France, March 1996.

[16] http://svmlight.joachims.org/.

[17] Mohammed J. Zaki. Efficient enumeration of frequent sequences. In *Proceedings of the 1998 ACM 7th International Conference on Information and Knowledge Management(CIKM'98)*, Washington, United States, November 1998.