ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/276462167

A novel method for constrained class association rule mining

Article *in* Information Sciences · November 2015

DOI: 10.1016/j.ins.2015.05.006

CITATIONS
8

READS
195

4 authors:

Dang Nguyen
Deakin University
14 PUBLICATIONS **50** CITATIONS
SEE PROFILE

Loan T.T. Nguyen
Nguyen Tat Thanh University
20 PUBLICATIONS **94** CITATIONS
SEE PROFILE

Bay Vo
Ho Chi Minh City University of Technology (H…
115 PUBLICATIONS **701** CITATIONS
SEE PROFILE

Tzung-Pei Hong
National University of Kaohsiung
653 PUBLICATIONS **5,856** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Advanced Conputationnal Methods for Knowledge Engineering View project

Novel algorithms for sequence mining, sequence prediction and utility pattern mining View project

All content following this page was uploaded by Loan T.T. Nguyen on 13 June 2015.

The user has requested enhancement of the downloaded file. All in-text references underlined in blue are added to the original document and are linked to publications on ResearchGate, letting you access and read them immediately.

Information Sciences 320 (2015) 107–125

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A novel method for constrained class association rule mining

Dang Nguyen [a,b], Loan T.T. Nguyen [c], Bay Vo [d,*], Tzung-Pei Hong [e,f]

[a] Division of Data Science, Ton Duc Thang University, Ho Chi Minh, Viet Nam
[b] Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh, Viet Nam
[c] Faculty of Information Technology, VOV College, Ho Chi Minh, Viet Nam
[d] Faculty of Information Technology, Ho Chi Minh City University of Technology, Viet Nam
[e] Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan, ROC
[f] Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, ROC

ARTICLE INFO

Article history:
Received 26 June 2013
Received in revised form 26 April 2015
Accepted 3 May 2015
Available online 7 May 2015

Keywords:
Associative classification
Class association rule
Data mining
Useful rules

ABSTRACT

To create a classifier using an associative classification algorithm, a complete set of class association rules (CARs) is obtained from the training dataset. Most generated rules, however, are either redundant or insignificant. They not only confuse end users during decision-making but also decrease the performance of the classification process. Thus, it is necessary to eliminate redundant or unimportant rules as much as possible before they are used. A related problem is the discovery of interesting or useful rules. In existing classification systems, the set of such rules may not be discovered easily. However, in real world applications, end users often consider the rules with consequences that contain one of particular classes. For example, in cancer screening applications, researchers are very interested in rules that classify genes into the "cancer" class. This paper proposes a novel approach for mining relevant CARs that considers constraints on the rule consequent. A tree structure for storing frequent itemsets from the dataset is designed. Then, some theorems for pruning tree nodes that cannot generate rules satisfying the class constraints are provided and proved. Finally, an efficient algorithm for mining constrained CARs is presented. Experiments show that the proposed method is faster than existing methods.

© 2015 Elsevier Inc. All rights reserved.

1. Introduction

Association rule mining and classification are two important and common problems in the data mining field. Therefore, numerous approaches have been proposed for the integration of these models. Examples include classification based on association rules (CBA) [23], a classification model based on multiple association rules [21], a classification model based on predictive association rules [41], multi-class and multi-label associative classification [34], a classifier based on maximum entropy [35], the use of an equivalence class rule tree [39], a lattice-based approach for classification [29], the integration of taxonomy information into classifier construction [6], the integration of classification rules into a neural network [20], a condition-based classifier with a small number of rules [11], an efficient classification approach with a rule quality metric [10], and a combination of a Netconf measure and a rule ordering strategy based on rule size [15].

The implementation processes of these approaches are often similar. Firstly, a complete set of class association rules (CARs) is mined from the training dataset. Then, a subset of CARs is selected to form the classifier. However, the complete

* Corresponding author at: Faculty of Information Technology, Ho Chi Minh City University of Technology, Viet Nam.
E-mail addresses: nguyenphamhaidang@tdt.edu.vn, nguyenphamhaidang@outlook.com (D. Nguyen), nguyenthithuyloan@vov.org.vn (L.T.T. Nguyen), bayvodinh@gmail.com (B. Vo), tphong@nuk.edu.tw (T.-P. Hong).

http://dx.doi.org/10.1016/j.ins.2015.05.006
0020-0255/© 2015 Elsevier Inc. All rights reserved.

set of CARs is often very large as it contains many redundant or insignificant rules. These useless rules not only waste storage space and decrease the performance of a classifier, but they also have a negative affect on decision-making.

To solve this problem, effort has been devoted to pruning redundant rules or ranking rules. The typical case of redundant rules is rule cover [36], where a rule is redundant if it is covered by others. The concept of sub-rules has been developed [23,29,39]. Another approach for pruning redundant rules to analyze the relations between rules with respect to objects was proposed by Liu et al. [24]. The authors used the Galois connections between objects and rules to determine the dependent relationships among rules. Their study showed that one rule is equivalent to another if they are supported by the same objects. Besides pruning strategies, some researchers have reported that ranking mechanisms are also important for CAR mining in associative classification since they directly affect classification accuracy. Therefore, several rule ranking approaches have been developed Li and Cercone [19] applied rough set theory to discover and rank significant rules Najeeb et al. [26] proposed a hybrid approach for rule ranking based on an evolutionary algorithm Cai et al. [8] ranked the interestingness of rules within an equivalent rule group for gene expression classification using two proposed interestingness measures called Max-Subrule-Conf and Min-Subrule-Conf Chen et al. [9] improved the performance of an associative classifier by rule prioritization. They proposed the MLRP algorithm, which re-ranks the execution order of CARs using rule priority to reduce the influence of rule dependence.

Although rule pruning and rule ranking approaches can help to eliminate redundant rules and obtain important rules, improving classifier performance, there has been little success with regard to discovering interesting or useful rules from an end user's point of view. In the real world, end users often consider the rules whose rule consequences contain one particular class. For example, while analyzing HIV/AIDS data, epidemiologists often concentrate on the rules whose rule consequences are HIV-Positive. Similarly, while mining banking data, bank loan officers often pay attention to the rules that classify a loan applicant into the "risky" class. Under this context, the present study proposes a novel and fast method for mining CARs with consideration of class constraints.

The contributions of this paper are stated as follows:

(1) A novel tree structure called the Novel Equivalence Class Rule tree (NECR-tree) is proposed for efficiently mining CARs with class constraints. Each node in NECR-tree contains attribute values and their information.
(2) Some theorems for quickly pruning nodes that are unable to generate rules satisfying the class constraints are developed.
(3) A novel and efficient algorithm for mining constrained CARs based on the provided theorems is presented.

The rest of this paper is organized as follows. In Section 2, some preliminary concepts of CAR mining are briefly given. Section 3 discusses work related to constrained association rule mining and CAR mining. The main contributions of the paper are presented in Section 4, in which the novel tree structure, called NECR-tree, is presented, and some theorems for eliminating unnecessary tree nodes are provided. The proposed algorithm for mining CARs with consideration of class constraints is also presented. Experimental results are discussed in Section 5. The conclusions and ideas for future work are given in Section 6.

2. Preliminary concepts

Let D be a dataset with d attributes $\{A_1, A_2, \ldots, A_d\}$ and n denote records (objects), where each record has an object identifier (*OID*). Let $C = \{c_1, c_2, \ldots, c_k\}$ be a list of class labels (k is the number of classes). Let *Constraint_Class* be a subset of C containing particular class labels considered by end users. A specific value of an attribute A_i and the mth record is denoted by a_{mi} ($m \in [1,n], i \in [1,d]$) and a specific value of class C is denoted by c_x ($x \in [1,k]$).

Definition 1. An *item* is described as an attribute and a specific value for that attribute, denoted by $\{(A_i, a_{mi})\}$ ($m \in [1,n], i \in [1,d]$).

Definition 2. An *itemset* is a set of *items*, denoted by $\{(A_i, a_{mi}), \ldots, (A_j, a_{mj})\}$ ($m \in [1,n], i,j \in [1,d]$, and $i \neq j$).

Definition 3. CAR R has the form $\{(A_i, a_{mi}), \ldots, (A_j, a_{mj})\} \rightarrow c_x$, where $\{(A_i, a_{mi}), \ldots, (A_j, a_{mj})\}$ is an *itemset* and $c_x \in C$ is a class label.

Definition 4. The actual occurrence $ActOcc(R)$ of rule R in D is the number of records of D that match R's antecedent.

Definition 5. The support of rule R, denoted by $Sup(R)$, is the number of records of D that match R's antecedent and R's consequent.

Definition 6. The confidence of rule R, denoted by $Conf(R)$, is defined as:

$$Conf(R) = \frac{Sup(R)}{ActOcc(R)}$$

Example 1. Table 1 shows a sample dataset that contains ten objects, three attributes (A, B, and C), and three classes (1, 2, and 3). For example, consider rule $R : \{(A, a1)\} \rightarrow 1$. We have $ActOcc(R) = 5$ and $Sup(R) = 3$ because there are five objects with $A = a1$, with three objects having the same class (i.e., 1). We also have $Conf(R) = \frac{Sup(R)}{ActOcc(R)} = \frac{3}{5}$.

3. Related work

This section briefly reviews the related literature.

3.1. Constrained association rule mining

Association rule mining finds all rules in a dataset that satisfy the minimum support (*minSup*) and the minimum confidence (*minConf*) thresholds. Association rule mining was introduced by Agrawal et al. [3]. The authors then proposed the Apriori algorithm [4], which has two main steps. Firstly, it generates all frequent itemsets from the dataset that satisfy *minSup*. Secondly, it generates association rules from the frequent itemsets that satisfy *minConf*. The weakness of this method is that it generates many candidates and scans the dataset many times, making it time-consuming. Thus, numerous approaches that reduce the mining time have been proposed. Some examples are Eclat [42,43], FP-Growth [14], BitTableFI [12], Index-BitTableFI [32], and dynamic bit vectors [38]. In the real world, end users are often interested in smaller sets of frequent itemsets and association rules that satisfy the given constraints. Post-processing requires a great deal of time and effort. Therefore, the problem of mining constrained frequent itemsets and association rules has been widely researched in the literature. Three main groups of strategies have been proposed for mining frequent itemsets with constraints. The first group, generate-and-test approaches, is mainly based on the Apriori algorithm and uses the level-wise approach to discover constrained frequent itemsets. MultipleJoins, Reorder, and Direct [33], CAP [27], CI-Miner [5], and CARGEMI [7] are some examples. Although these algorithms can use the properties of the constraints effectively, they have some disadvantages inherited from Apriori-based algorithms, including that they require multiple dataset scans that are as numerous as the longest frequent itemsets. The second group, divide-and-conquer approaches, adopts the divide-and-conquer methodology and uses compact data structures extended from the frequent-pattern (FP) tree to find constrained frequent itemsets. Some examples are FIC^A, FIC^M [31], FPS [18], and MCFPTree [22]. The third group, lattice-based approaches, applies lattice theory for mining frequent itemsets with constraints. Eclat2 [25], C-Charm [37], and MFS-Contain-IC [13] belong to this group.

These three approaches for mining constrained frequent itemsets cannot be applied to CAR mining with class constraints since they apply the constraints to itemsets only and do not generate constrained CARs directly. Additionally, to calculate the confidence of association rules, algorithms for mining constrained frequent itemsets have to scan the original dataset again to determine the support of rule antecedents. Because frequent itemsets in the rule antecedents do not contain the constrained itemsets, their support cannot be directly determined. Consequently, the problem of mining CARs with class constraints requires a different strategy.

3.2. CAR-Miner+ algorithm

Similar to association rule mining, CAR mining is the discovery of all CARs that satisfy *minSup* and *minConf*. However, the target of association rule mining is not pre-determined whereas that of CAR mining is (i.e., the class). The first method for mining CARs was proposed by [23], but it is time-consuming because of its generate-and-test mechanism Vo and Le [39] proposed a novel method for mining CARs using the Equivalence Class Rule tree (ECR-tree). An efficient algorithm, called ECR-CARM, was also proposed in their paper. ECR-CARM scans the dataset once and uses a set of object identifiers to determine the support of itemsets quickly. A tree structure for fast mining CARs was also developed. However, a lot of effort and

Table 1
Example dataset.

OID	A	B	C	Class
0	a1	b1	c1	1
1	a1	b1	c1	1
2	a1	b1	c1	1
3	a1	b1	c1	2
4	a1	b1	c1	2
5	a3	b2	c1	3
6	a2	b2	c2	2
7	a2	b2	c2	2
8	a3	b2	c2	3
9	a3	b2	c2	2

time are required for the candidate generate-and-test process because each node in the tree contains all values of one attribute Nguyen et al. [30] modified the ECR-tree structure to speed up the mining time. In their enhanced tree, called MECR-tree, each node contains only one value of an attribute instead of those of the whole group. Moreover, they also provided some theorems to identify the support for the child nodes and to prune unnecessary nodes quickly. Based on MECR-tree and these theorems, they proposed the CAR-Miner algorithm for mining CARs. Recently, [28] developed three parallel algorithms for mining CARs on a multi-core processor architecture. Other approaches for CAR mining have been reviewed in [2].

However, none of the above methods can be applied directly for mining constrained CARs. To deal with this requirement, an extended version of CAR-Miner called CAR-Miner+ is proposed as follows. Firstly, CAR-Miner is used to discover all CARs from the dataset. Then, post-processing is conducted to filter out rules that do not satisfy the constraints. The pseudo code of the CAR-Miner+ algorithm is shown in Fig. 1.

For details on CAR-Miner, please refer to the study of Nguyen et al. [30]. CAR-Miner+ is a straightforward extension of the original CAR-Miner algorithm and is easy to implement, but it fails to exploit the properties of the constraints. As a result, the main drawback of this approach lies in its computational complexity. Hence, the method proposed here attempts to push the class constraints as deep "inside" the computation as possible. The most noticeable improvement is that rather than building all tree nodes, which leads to a high computational cost, only nodes that can generate rules satisfying the class constraints are formed. As a result, the proposed approach is faster.

4. Mining constrained class association rules

4.1. Tree structure

This study proposes a novel tree structure, called NECR-tree, in which each node contains one *itemset* along with the following information:

(a) $(Obidset_1, Obidset_2, \ldots, Obidset_k)$: each $Obidset_i$ is a set of object identifiers (*OID*) that contain both *itemset* and class c_i. Note that k is the number of classes in the dataset.
(b) *pos*: stores the position of the class with the maximum cardinality of $Obidset_i$, i.e., $pos = \text{argmax}_{i\in[1,k]}\{|\ Obidset_i\ |\}$.
(c) *total*: stores the sum of cardinalities of all $Obidset_i$, i.e., $total = \sum_{i=1}^{k} |\ Obidset_i\ |$.

MECR-tree stores a list of $\#c_i$, which is the number of objects in *Obidset* that match class c_i. Hence, it is necessary to compute this list for all nodes. In contrast, NECR-tree does not store the list of $\#c_i$, reducing both memory usage and execution time. *pos* information is stored in NECR-tree. However, by using Theorem 2 (see Section 4.2), this information can be directly identified for many more nodes compared to those for CAR-Miner. In addition, unlike ECR-tree and MECR-tree, NECR-tree does not store the entire *Obidset*; instead, it partitions *Obidset* into a list of $Obidset_i$ to speed up the execution time of joining two nodes.

In NECR-tree, *itemset* is converted into the form $att \times values$ for easily programming, where:

(1) *att*: a list of attributes.
(2) *values*: a list of values, each of which is contained in one attribute in *att*.

Input: Dataset D, *minSup*, *minConf*, and *Constraint_Class*

Output: All CARs satisfying *minSup*, *minConf*, and *Constraint_Class*

1. $CARs$ = **CAR-Miner**(L_r, *minSup*, *minConf*);
2. Constraint_CARs = filterRules(*CARs*, *Constraint_Class*);

Procedure:

filterRules(*CARs*, *Constraint_Class*)

3. Constraint_CARs = $\varnothing$;
4. for each $rule \in CARs$ do
5. if $rule.consequent \in Constraint\_Class$ then
6. Constraint_CARs = Constraint_CARs $\cup$ $rule$;

Fig. 1. CAR-Miner+ algorithm.

Input: Dataset D, *minSup*, *minConf*, and *Constraint_Class*

Output: All CARs satisfying *minSup*, *minConf*, and *Constraint_Class*

Procedure:

Constraint-CAR-Miner(L_r, *minSup*, *minConf*, *Constraint_Class*)

1. CARs=$\emptyset$;
2. for all $l_i \in L_r$.children do
3. $iChild = 0$;
4. ENUMERATE-CAR(l_i, *minConf*, *Constraint_Class*);
5. $P_i = \emptyset$;
6. for all $l_j \in L_r$.children , with $j > i$ do
7. if $l_i.att \neq l_j.att$ then // using Theorem 1
8. $O.att = l_i.att \cup l_j.att$; // using bitwise operation
9. $O.values = l_i.values \cup l_j.values$;
10. $O.Obidset_i = l_i.Obidset_i \cap l_j.Obidset_i$;
11. if $\left|O.Obidset_{l_i.pos}\right| = \left|l_i.Obidset_{l_i.pos}\right|$ // using Theorem 2
12. $O.pos = l_i.pos$;
13. else if $\left|O.Obidset_{l_j.pos}\right| = \left|l_j.Obidset_{l_j.pos}\right|$ // using Theorem 2
14. $O.pos = l_j.pos$;
15. else
16. $O.pos = \text{argmax}_{i\in[1,k]}\left\{\left|O.Obidset_i\right|\right\}$;
17. $O.total = \sum_{i=1}^{k}\left|O.Obidset_i\right|$;
18. if $\left|O.Obidset_{O.pos}\right| \geq minSup$ then // using Corollary 1
19. if $\left|O.Obidset_t\right| \geq minSup$, with $\forall t \in Constraint\_Class$ then // using Theorem 3
20. $P_i = P_i \cup O$;
// using Theorem 4
21. if $O.pos = l_i.pos \notin Constraint\_Class$ and $\left|O.Obidset_{O.pos}\right| = \left|l_i.Obidset_{l_i.pos}\right|$ then
22. $iChild++$;
23. if $iChild = \left|P_i\right|$ then // using Theorem 4
24. $P_i = \emptyset$;
25. Constraint-CAR-Miner(P_i, *minSup*, *minConf*, *Constraint_Class*);

ENUMERATE-CAR(l, *minConf*, *Constraint_Class*)

26. conf = $\left|l.Obidset_{l.pos}\right| / l.total$;
27. if conf$\geq$*minConf* and $l.pos \in Constraint\_Class$ then // using Lemma 1
28. CARs=CARs$\cup\left\{l.itemset \rightarrow c_{pos}\left(\left|l.Obidset_{l.pos}\right|, \text{conf}\right)\right\}$;

Fig. 2. Proposed algorithm for mining CARs with class constraints.

Definition 7. Each node in NECR-tree is the following 5-tuple:

$\langle att, values, (Obidset_1, \ldots, Obidset_k), pos, total\rangle$

Example 2. Itemset $X = \{(A, a2), (B, b2)\}$ is denoted as $X = 3 \times a2b2$. A bit representation is used for itemset attributes. Attributes AB can be presented by 11 in this bit representation, so the value of these attributes is 3. Bitwise operations are then used to join itemsets quickly.

Example 3. In Table 1, *itemset* $X = \{(A, a2), (B, b2)\}$ is contained in objects 6 and 7, both of which belong to class 2. Therefore, node $\{(A, a2), (B, b2)\}(\varnothing, \underline{67}, \varnothing)$ (or $3 \times a2b2(\varnothing, \underline{67}, \varnothing)$) would be added to the NECR-tree if *minSup* were 2. This node has $Obidset_1 = \varnothing$ (i.e., no objects contain both itemset X and class 1), $Obidset_2 = \{6, 7\}$ (or $Obidset_2 = 67$ for short) (i.e., objects 6 and 7 contain both itemset X and class 2), $Obidset_3 = \varnothing$ (i.e., no objects contain both itemset X and class 3), $pos = 2$ (a line under position 2 of list $Obidset_i$), and $total = 2$. The *pos* value is 2 because the count of *Obidset* for class 2 is the maximum.

To remove redundant rules, if multiple rules are generated from a given node that satisfies *minConf*, we would select the rule R with the highest confidence. This means that the rule R has the form $itemset \rightarrow c_{pos}$ with $Sup(R) = \mid Obidset_{pos} \mid$ and $Conf(R) = \frac{|Obidset_{pos}|}{total}$.

Example 4. Assume that *minSup* is 2 and *minConf* is 0.4. Node $1 \times a1(\underline{012}, 34, \varnothing)$ has two rules, namely $R1 : a1 \rightarrow 1$ and $R2 : a1 \rightarrow 2$, that satisfy *minConf*. $R1$ is added to the final rule set since $Conf(R1) = 3/5 > Conf(R2) = 2/5$.

4.2. Proposed algorithm

In this section, some theorems for pruning unnecessary nodes are developed. Then, an efficient algorithm for mining CARs with class constraints based on the provided theorems is proposed.

Theorem 1. *Given two nodes X and Y, if* $X.att = Y.att$ *and* $X.values \neq Y.values$, *then* $X.Obidset_i \cap Y.Obidset_i = \varnothing$ $(\forall i \in [1, k])$.

Proof. Because $X.att = Y.att$ and $X.values \neq Y.values$, there exist a $val_1 \in X.values$ and a $val_2 \in Y.values$ such that val_1 and val_2 belong to the same attributes but different values. If an object identifier OID_j contains val_1, it cannot include val_2. Thus, $\forall OID_j \in X.Obidset_i$. Then, it can be inferred that $OID_j \notin Y.Obidset_i$ and $X.Obidset_i \cap Y.Obidset_i = \varnothing$ $(j \in [1, n], i \in [1, k])$. □

Theorem 1 infers that if two itemsets X and Y have the same attributes, it is unnecessary to combine them to itemset XY since $Sup(XY) = 0$.

Example 5. Consider two nodes $1 \times a1(\underline{012}, 34, \varnothing)$ and $1 \times a2(\varnothing, \underline{67}, \varnothing)$, whose attribute is $att = 1$. It can be seen that $Obidset_i(a1a2) = Obidset_i(a1) \cap Obidset_i(a2) = \varnothing$. Similarly, $3 \times a1b1(\underline{012}, 34, \varnothing) \cap 3 \times a2b2(\varnothing, \underline{67}, \varnothing) = \varnothing$ since they have the same attributes AB but different values ($a1b1$ and $a2b2$).

Theorem 2. *Given two nodes X and Y, if X is a parent node of Y and* $\mid Y.Obidset_{X.pos} \mid = \mid X.Obidset_{X.pos} \mid$, *then* $Y.pos = X.pos$.

Proof. X is a parent node of Y, so $Y.Obidset_i = X.Obidset_i \cap Z.Obidset_i$, in which Z is another parent node of Y, meaning that $\mid Y.Obidset_i \mid \leqslant \mid X.Obidset_i \mid$ (1). Based on the definition of *pos*, we have $\mid X.Obidset_i \mid \leqslant \mid X.Obidset_{X.pos} \mid$ $(\forall i \in [1, k])$ (2). Regarding the assumption, we also have $\mid X.Obidset_{X.pos} \mid = \mid Y.Obidset_{X.pos} \mid$ (3). From (2) and (3), we have $\mid X.Obidset_i \mid \leqslant \mid Y.Obidset_{X.pos} \mid$ (4). From (1) and (4), it can be inferred that $\mid Y.Obidset_i \mid \leqslant \mid Y.Obidset_{X.pos} \mid$ $(\forall i \in [1, k])$. It can thus be concluded that $\mid Y.Obidset_{X.pos} \mid$ is maximum and $Y.pos = X.pos$. □

Based on Theorem 2, when two parent nodes are joined into one child node, then the itemset of the child node is always a superset of the itemset of each parent node. We compare the cardinality of *Obidset* at position *pos* of each parent node with the cardinality of *Obidset* at the same position as that of the child node. If they are equal, the position *pos* for the child node does not need to be calculated since it is the same as that of its parent node.

Example 6. Node $Y = 6 \times b2c2(\varnothing, 679, 8)$ has two parent nodes $X = 2 \times b2(\varnothing, \underline{679}, 58)$ and $Z = 4 \times c2(\varnothing, \underline{679}, 8)$. $Y.pos = X.pos = 2$ since $\mid Y.Obidset_2 \mid = \mid 679 \mid = 3 = \mid X.Obidset_2 \mid$.

Compared with Theorem 2 in the study of Nguyen et al. [30], our Theorem 2 directly identifies position *pos* for many more nodes. Assume that node $Y = 3 \times a3b3(46, \varnothing)$ has two parent nodes $X = 1 \times a3(\underline{46}, 5)$ and $Z = 2 \times b3(\underline{467}, \varnothing)$. By using Theorem 2 in the study of Nguyen et al. [30], we cannot identify $Y.pos$ directly because $\mid Y.Obidset \mid = \mid 46 \mid = 2 \neq \mid X.Obidset \mid = \mid 465 \mid = 3$ and $\mid Y.Obidset \mid = 2 \neq \mid Z.Obidset \mid = 3$. However, by using our Theorem 2, we can determine $Y.pos = 1$.

Lemma 1. *Node X can generate rule R that satisfies minSup and minConf if and only if* $|X.Obidset_{X.pos}| \geqslant minSup$ *and* $\frac{|X.Obidset_{X.pos}|}{X.total} \geqslant minConf$.

Proof. Suppose that $|X.Obidset_{X.pos}| < minSup$ and rule R satisfies *minSup*. This implies that $|X.Obidset_i| < minSup$. Thus, $Sup(R) < minSup$. This result is in conflict with the supposition.

We select only the rule with the highest confidence, so $Conf(R) = \frac{|X.Obidset_{X.pos}|}{X.total} \geqslant minConf$. □

Corollary 1. *The condition for adding one node to the NECR-tree is* $|Obidset_{pos}| \geqslant minSup$.

Proof. From Lemma 1, this condition ensures that the node can generate a rule satisfying *minSup*. □

Theorem 3. *Given node X, this node and its child nodes cannot generate rules satisfying Constraint_Class if* $|X.Obidset_i| < minSup$ $(\forall i \in Constraint\_Class)$.

Proof. Let Y be a child node of X. Then, we have $|Y.Obidset_i| \leqslant |X.Obidset_i| < minSup$ $(\forall i \in Constraint\_Class)$. From Lemma 1, if Y can generate rule R_2 that satisfies *minSup*, then $|Y.Obidset_{Y.pos}| \geqslant minSup$. This implies that $Y.pos \neq i$ $(\forall i \in Constraint\_Class)$. Thus, rule R_2 has the form $Y.itemset \rightarrow Y.pos$ and cannot satisfy *Constraint_Class*. Here, we converted c_{pos} to *pos* and $att \times values$ to *itemset* for ease of representation. Similarly, X also cannot generate a rule satisfying *Constraint_Class*. □

From Corollary 1, we only add nodes with $|Obidset_{pos}| \geqslant minSup$ to the NECR-tree. However, regarding Theorem 3, if these nodes have $|Obidset_i| < minSup$ $(\forall i \in Constraint\_Class)$, then they are not attached to the tree.

Example 7. With $minSup = 2$ and $Constraint\_Class = \{2\}$, consider node $1 \times a3(\varnothing, 9, \underline{58})$. Regarding Corollary 1, this node is added to the tree since $|Obidset_3| \geqslant minSup$. However, this node with $|Obidset_2| = |9| < minSup$ $(2 \in Constraint\_Class)$ is not attached to the tree according to Theorem 3.

Theorem 4. *Given node X, if this node cannot generate rules satisfying Constraint_Class, and its child nodes* Y_j *have* $X.pos = Y_j.pos$ *and* $|X.Obidset_{X.pos}| = |Y_j.Obidset_{Y_j.pos}|$, *then its grandchild nodes also cannot generate rules satisfying Constraint_Class.*

Proof. Because node X and nodes Y_j do not generate rules satisfying *Constraint_Class*, $X.pos \notin Constraint\_Class$ and $Y_j.pos \notin Constraint\_Class$. Besides, because $X.pos = Y_j.pos$ and $|Y_j.Obidset_{Y_j.pos}| \leqslant |X.Obidset_{X.pos}|$, $Y_j.Obidset_{Y_j.pos} = X.Obidset_{X.pos}$. Hence, the grandchild nodes also have $pos \notin Constraint\_Class$. This means that they cannot generate rules that satisfy *Constraint_Class*. □

From Theorem 4, if the parent node and the child nodes cannot generate constrained rules, and they have the same position *pos* and the same *Obidset* at position *pos*, then we can remove them from the tree because their grandchild nodes also cannot generate constrained rules.

Example 8. Node $1 \times a1(\underline{012}, 34, \varnothing)$ has two child nodes, namely $3 \times a1b1(\underline{012}, 34, \varnothing)$ and $5 \times a1c1(\underline{012}, 34, \varnothing)$. Suppose that $minSup = 2$ and $Constraint\_Class = \{2\}$. Three nodes are added to the tree because $|Obidset_2| \geqslant minSup$ (from Theorem 3). However, they and their descendants cannot generate constrained rules. Based on Theorem 4, they are removed from the tree.

Based on these four theorems, we develop an efficient algorithm for mining constrained CARs. By Theorem 1, we do not need to join two nodes with the same attributes, and by Theorem 2, we do not need to compute the information for some child nodes. By Theorems 3 and 4, we do not need to generate some nodes. The proposed algorithm is outlined in Fig. 2.

First, the root node of the NECR-tree (L_r) contains child nodes, each of which contains a single frequent 1-itemset. It is noted that the nodes that cannot generate constrained rules as determined from Theorem 3 are removed from L_r. Then, the procedure Constraint-CAR-Miner is called with the parameters L_r, *minSup*, *minConf*, and *Constraint_Class* to mine all constrained CARs from dataset D.

The Constraint-CAR-Miner procedure (Fig. 2 considers each node l_i with all the other nodes l_j in L_r, with $j > i$ (lines 2 and 6), to generate a candidate child node O. With each pair (l_i, l_j), the algorithm checks whether $l_i.att \neq l_j.att$ (line 7, using Theorem 1. If they are different, it computes the elements $att, values, Obidset_i,$ and *total* for the new node O (lines 8–10 and 17). Line 11 checks whether the number of *OIDs* at position $l_i.pos$ of node l_i is equal to the number of *OIDs* at the same position $l_i.pos$ of node O (by Theorem 2. If this is true, the algorithm copies the *pos* information from node l_i to node O (line 12). Similarly, in the event of a false result on line 11, the algorithm does the same check between l_j and O (line 13) and it

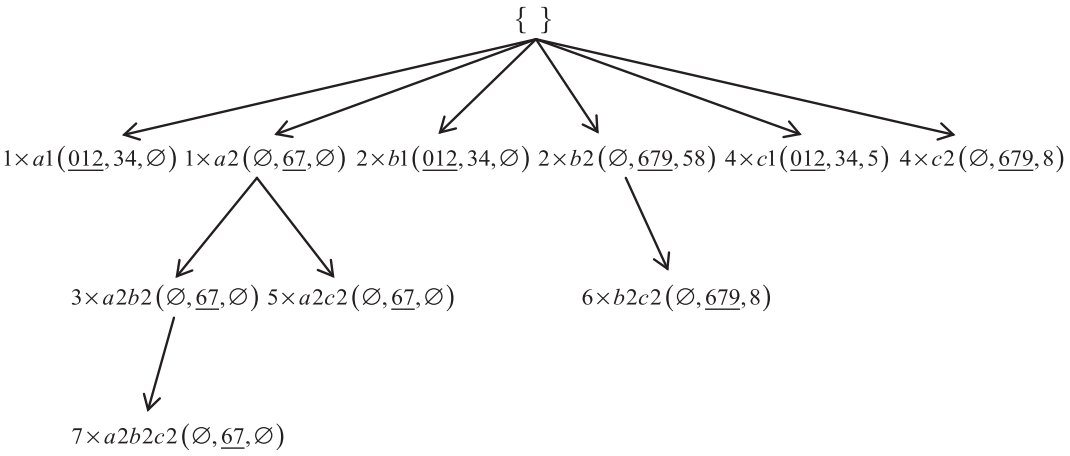

Fig. 3. NECR-tree for dataset in Table 1.

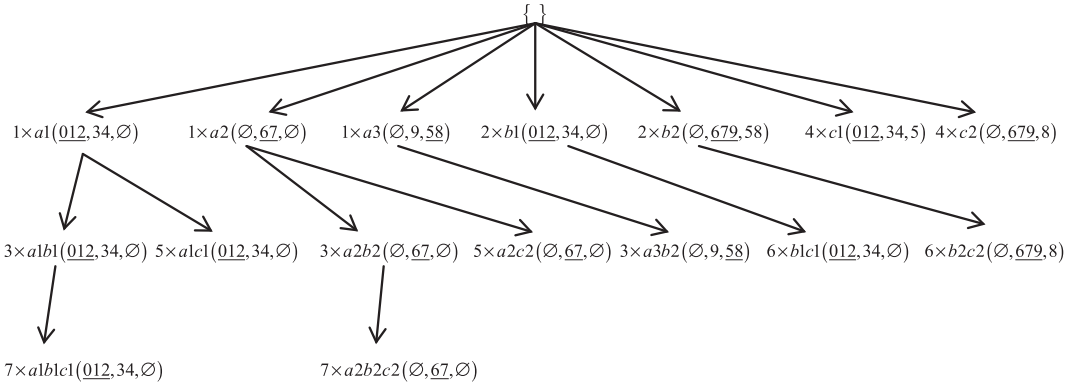

Fig. 4. Whole tree generated by CAR-Miner+ for dataset in Table 1.

copies the *pos* information from node l_j to node O if the check holds true (line 14). Otherwise, the algorithm computes $O.pos$ by calculating the max value of $|\ O.Obidset_i\ |$ (line 16). After computing all information for node O, the algorithm uses Corollary 1 and Theorem 3 to check whether this node can generate a rule that satisfies *minSup* (line 18) and *Constraint_Class* (line 19), respectively. Then, it adds node O to P_i (P_i is initialized empty on line 5) if both conditions hold true (line 20). On line 21, node O is assessed with Theorem 4 (line 21) and variable *iChild* is increased (line 22) if O is a case of Theorem 4. On lines 23–24, if all child nodes generated from parent node l_i satisfy Theorem 4, then they are removed from P_i since they and their descendant nodes cannot generate rules satisfying *Constraint_Class*. Finally, Constraint-CAR-Miner is recursively called with a new set P_i as its input parameter (line 25).

The procedure ENUMERATE-CAR(l, *minConf*, *Constraint_Class*) generates a rule from node l. It firstly computes the confidence of the rule (line 26); if the confidence of this rule satisfies *minConf* (by Lemma 1), and the rule consequent matches the constraints (line 27), then this rule is added to the set of CARs (line 28).

4.3. An illustrative example

In this section, the example dataset in Table 1 is used to illustrate the process of Constraint-CAR-Miner with *minSup* = 1.6, *minConf* = 0.6, and *Constraint_Class*={2}. Fig. 3 shows the results of this process.

The NECR-tree is built from the dataset in Table 1 as follows. At the beginning, all frequent 1-itemsets are identified: $\{1\times a1(\underline{012},34,\varnothing),1\times a2(\varnothing,\underline{67},\varnothing),1\times a3(\varnothing,9,\underline{58}),2\times b1(\underline{012},34,\varnothing),2\times b2(\varnothing,\underline{679},58),4\times c1(\underline{012},34,5),4\times c2(\varnothing,\underline{679},8)\}$. However, only frequent 1-itemsets satisfying Theorem 3 are added to the root node L_r.

Therefore, L_r contains $\{1\times a1(\underline{012},34,\varnothing),1\times a2(\varnothing,\underline{67},\varnothing),2\times b1(\underline{012},34,\varnothing),2\times b2(\varnothing,\underline{679},58),4\times c1(\underline{012},34,5),4\times c2(\varnothing,\underline{679},8)\}$. Then, procedure Constraint-CAR-Miner is called with parameter L_r. We use node $l_i = 1\times a1(\underline{012},34,\varnothing)$ as an example to illustrate the process of Constraint-CAR-Miner. l_i is joined with all nodes following it in L_r:

Table 2
Characteristics of experimental datasets.

Dataset	# of attributes	# of classes	# of distinctive values	# of objects
Breast	12	2	737	699
Lymph	18	4	63	148
Vehicle	19	4	1434	846
German	21	2	1077	1000
Ionosphere	35	2	209	351
Chess	37	2	76	3196
Pumsb	74	5	2113	49,046
Connect-4	43	3	130	67,557

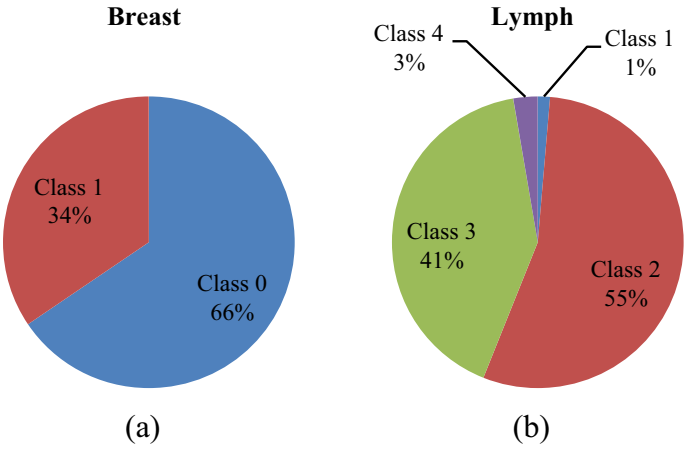

Fig. 5. Distribution of each class in (a) Breast and (b) Lymph datasets.

- With node $l_j = 1 \times a2(\varnothing, \underline{67}, \varnothing)$: they ($l_i$ and l_j) have the same attributes but different values. We do nothing with them.
- With node $l_j = 2 \times b1(\underline{012}, 34, \varnothing)$: because their attributes are different, five elements are computed:
  (1) $O.att = l_i.att \cup l_j.att = 1 \mid 2 = 3$ or 11 in bit representation.
  (2) $O.values = l_i.values \cup l_j.values = a1 \cup b1 = a1b1$.
  (3) $O.Obidset_i = l_i.Obidset_i \cap l_j.Obidset_i = (\underline{012}, 34, \varnothing) \cap (\underline{012}, 34, \varnothing) = (\underline{012}, 34, \varnothing)$.
  (4) Since $\mid O.Obidset_1 \mid = \mid l_i.Obidset_1 \mid, O.pos = 1$ (by Theorem 2).
  (5) $O.total = 5$.
  $\mid O.Obidset_{pos} \mid = 3 \geqslant minSup$ and $\mid O.Obidset_2 \mid = 2 \geqslant minSup$, so O is added to P_i (by Corollary 1 and Theorem 3, respectively). Therefore, $P_i = \{3 \times a1b1(\underline{012}, 34, \varnothing)\}$ and $iChild = 1$ (by Theorem 4).
- With node $l_j = 2 \times b2(\varnothing, \underline{679}, 58)$: because their attributes are different, five elements are computed:
  (1) $O.att = l_i.att \cup l_j.att = 1 \mid 2 = 3$ or 11 in bit representation.
  (2) $O.values = l_i.values \cup l_j.values = a1 \cup b2 = a1b2$.
  (3) $O.Obidset_i = l_i.Obidset_i \cap l_j.Obidset_i = (\underline{012}, 34, \varnothing) \cap (\varnothing, \underline{679}, 58) = \varnothing$.
  (4) $O.pos = 1$.
  (5) $O.total = 0$.
  Since $\mid O.Obidset_{pos} \mid = 0 < minSup$, O is not added to P_i (by Corollary 1.
- With node $l_j = 4 \times c1(\underline{012}, 34, 5)$: because their attributes are different, five elements are computed:
  (1) $O.att = l_i.att \cup l_j.att = 1 \mid 4 = 5$ or 101 in bit representation.
  (2) $O.values = l_i.values \cup l_j.values = a1 \cup c1 = a1c1$.
  (3) $O.Obidset_i = l_i.Obidset_i \cap l_j.Obidset_i = (\underline{012}, 34, \varnothing) \cap (\underline{012}, 34, 5) = (\underline{012}, 34, \varnothing)$.
  (4) Since $\mid O.Obidset_1 \mid = \mid l_i.Obidset_1 \mid, O.pos = 1$ (by Theorem 2).
  (5) $O.total = 5$.
  $\mid O.Obidset_{pos} \mid = 3 \geqslant minSup$ and $\mid O.Obidset_2 \mid = 2 \geqslant minSup$, so O is added to P_i.
  Therefore, $P_i = \{3 \times a1b1(\underline{012}, 34, \varnothing), 5 \times a1c1(\underline{012}, 34, \varnothing)\}$ and $iChild = 2$.
- With node $l_j = 4 \times c2(\varnothing, \underline{679}, 8)$: because their attributes are different, five elements are computed:
  (1) $O.att = l_i.att \cup l_j.att = 1 \mid 4 = 5$ or 101 in bit representation.
  (2) $O.values = l_i.values \cup l_j.values = a1 \cup c2 = a1c2$.
  (3) $O.Obidset_i = l_i.Obidset_i \cap l_j.Obidset_i = (\underline{012}, 34, \varnothing) \cap (\varnothing, \underline{679}, 8) = \varnothing$.

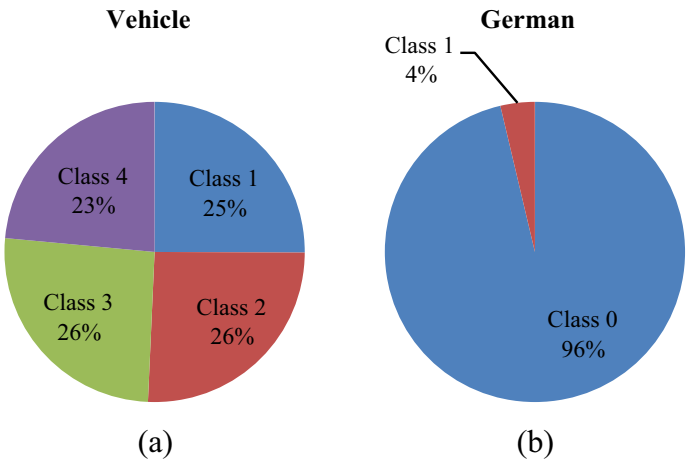

Fig. 6. Distribution of each class in (a) Vehicle and (b) German datasets.

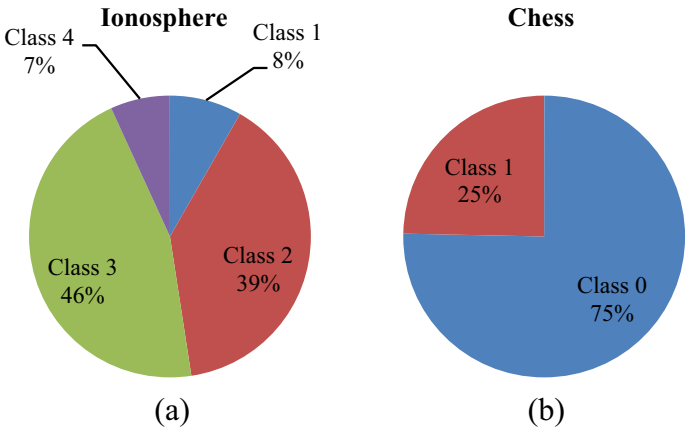

Fig. 7. Distribution of each class in (a) Ionosphere and (b) Chess datasets.

(4) $O.pos = 1$.
(5) $O.total = 0$.

Since $|\ O.Obidset_{pos}\ | = 0 < minSup$, O is not added to P_i.

Once P_i has been created, Constraint-CAR-Miner is called recursively with parameters P_i, *minSup*, *minConf*, and *Constraint_Class*. However, before the procedure is called, the algorithm checks whether all nodes in P_i match Theorem 4. If this is true, we remove all of them from P_i. For example, consider $P_i = \{3 \times a1b1(\underline{012}, 34, \varnothing), 5 \times a1c1(\underline{012}, 34, \varnothing)\}$ and $iChild = 2$. Condition $iChild = |\ P_i\ |$ holds true. Thus, we remove all nodes in P_i and do not need to call the Constraint-CAR-Miner procedure.

Similarly, with $P_i = \{3 \times a2b2(\varnothing, \underline{67}, \varnothing), 5 \times a2c2(\varnothing, \underline{67}, \varnothing)\}$, consider the process to make the child nodes of $l_i = 3 \times a2b2(\varnothing, \underline{67}, \varnothing)$:

- With node $l_j = 5 \times a2c2(\varnothing, \underline{67}, \varnothing)$: because their attributes are different, five elements are computed:
  (1) $O.att = l_i.att \cup l_j.att = 3 \mid 5 = 7$ or 111 in bit representation.
  (2) $O.values = l_i.values \cup l_j.values = a2b2 \cup a2c2 = a2b2c2$.
  (3) $O.Obidset_i = l_i.Obidset_i \cap l_j.Obidset_i = (\varnothing, \underline{67}, \varnothing) \cap (\varnothing, \underline{67}, \varnothing) = (\varnothing, \underline{67}, \varnothing)$.
  (4) Since $|\ O.Obidset_2\ | = |\ l_i.Obidset_2\ |, O.pos = 2$.
  (5) $O.total = 2$.
  $|\ O.Obidset_{pos}\ | = 2 \geqslant minSup$ and $|\ O.Obidset_2\ | = 2 \geqslant minSup$, so O is added to P_i. Therefore, $P_i = \{7 \times a2b2c2(\varnothing, \underline{67}, \varnothing)\}$ and $iChild = 0$.

Constrained rules are easily generated while traversing node l_i (Line 4) by calling procedure ENUMERATE-CAR(l_i, *minConf*, *Constraint_Class*). For example, while traversing node $l_i = 1 \times a2(\varnothing, \underline{67}, \varnothing)$, the procedure computes the confidence of the candidate rule (line 26), $\text{conf} = |\ l_i.Obidset_{l_i.pos}\ | / l_i.total = 2/2 = 1$. $\text{conf} \geqslant minConf$ and $l_i.pos \in Constraint\_Class$, so rule

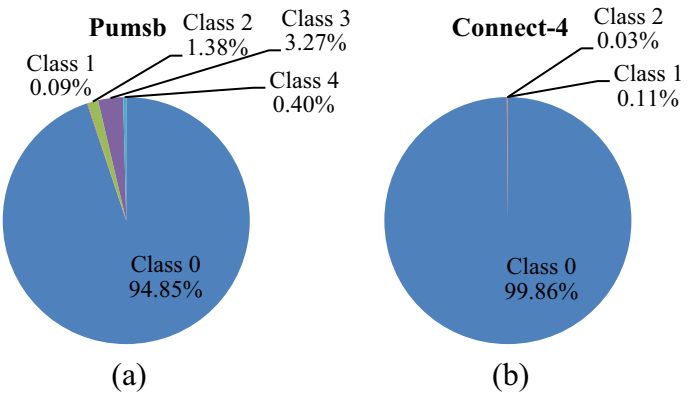

Fig. 8. Distribution of each class in (a) Pumsb and (b) Connect-4 datasets.

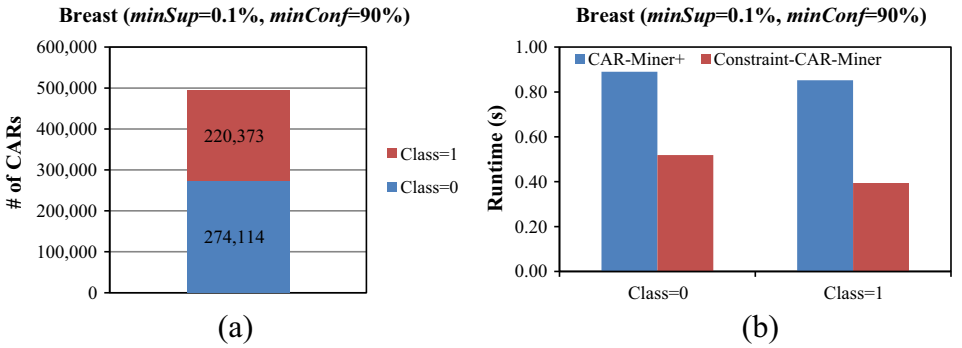

Fig. 9. (a) Number of constrained CARs for each class and (b) runtimes of CAR-Miner+ and Constraint-CAR-Miner for Breast dataset.

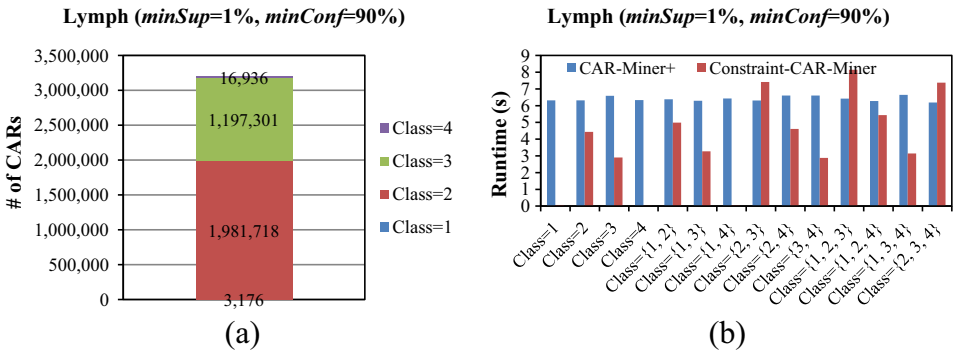

Fig. 10. (a) Number of constrained CARs for each class and (b) runtimes of CAR-Miner+ and Constraint-CAR-Miner for Lymph dataset.

$\{(A, a2)\} \rightarrow 2(2,1)$ is added to the set of CARs. The meaning of this rule is "If A = a2, then class = 2" (support = 2 and confidence = 1).

By using Theorems 3 and 4, the proposed algorithm does not need to generate some tree nodes, such as $\{1 \times a3(\varnothing, 9, \underline{58}), 3 \times a1b1(\underline{012}, 34, \varnothing), 5 \times a1c1(\underline{012}, 34, \varnothing), 3 \times a3b2(\varnothing, 9, \underline{58}), 6 \times b1c1(\underline{012}, 34, \varnothing), 7 \times a1b1c1(\underline{012}, 34, \varnothing)\}$, whereas CAR-Miner+ generates the whole tree, as shown in Fig. 4.

5. Experiments

Experiments were conducted to show the efficiency of the proposed algorithm. The algorithms were coded in C# using Microsoft Visual Studio.NET 2010 Express on a computer with an Intel Core i7-2600 3.40-GHz CPU and 4.00 GB of RAM running Windows 7 Enterprise (64-bit) SP1.

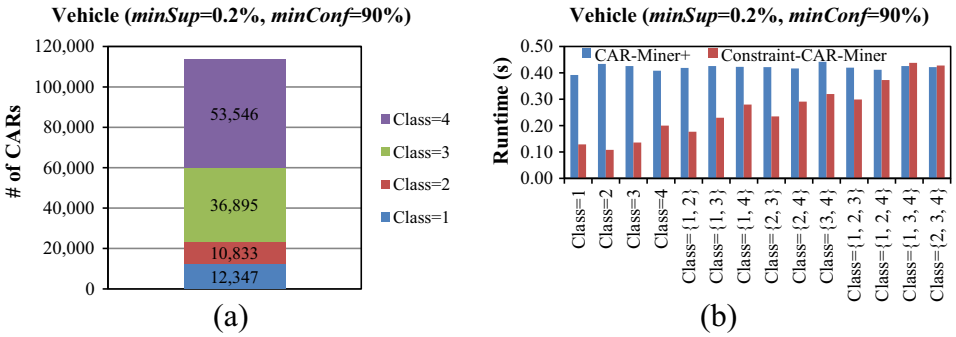

Fig. 11. (a) Number of constrained CARs for each class and (a) runtimes of CAR-Miner+ and Constraint-CAR-Miner for Vehicle dataset.

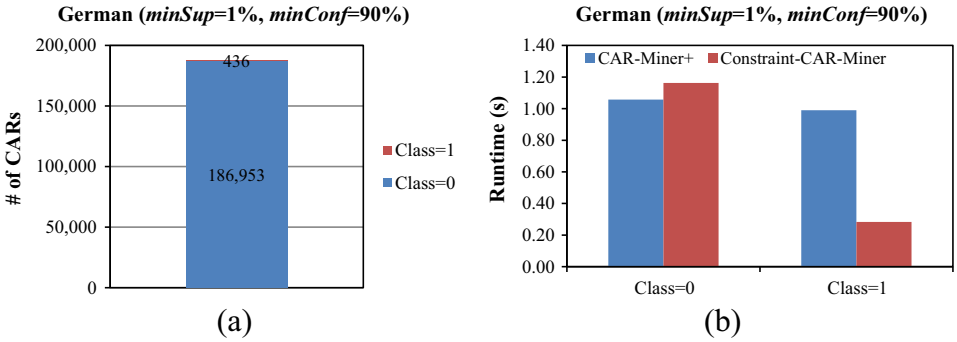

Fig. 12. (a) Number of constrained CARs for each class and (b) runtimes of CAR-Miner+ and Constraint-CAR-Miner for German dataset.

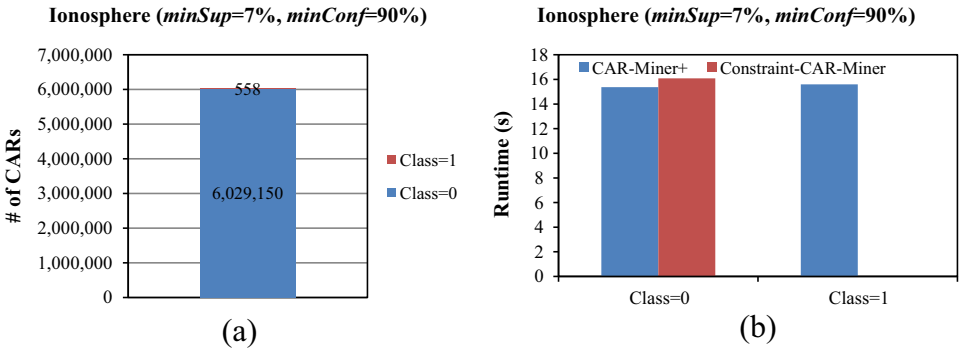

Fig. 13. (a) Number of constrained CARs for each class and (b) runtimes of CAR-Miner+ and Constraint-CAR-Miner for Ionosphere dataset.

5.1. Characteristics of experimental datasets

The experiments were tested with datasets obtained from the UCI Machine Learning Repository (http://mlearn.ics.uci.edu). Table 2 shows the main characteristics of the experimental datasets, namely the number of attributes (including the class attribute), the number of class labels, the number of distinctive values (i.e., the total number of distinct values in all attributes), and the number of objects (or records) in each dataset. The experimental datasets have different features. The Connect-4, Chess, and Pumsb datasets have many attributes, distinctive values, and objects (records), whereas the other datasets have few attributes and objects.

Figs. 5–8 show the distribution of each class in each dataset. It is clear that some datasets (Breast, Vehicle, and Chess) have quite balanced distributions for each class whereas other datasets (Lymph, German, Ionosphere, Pumsb, and Connect-4) have significant disproportions in their class distribution. For instance, Classes 0 and 1 account for 96% and only 4% in German,

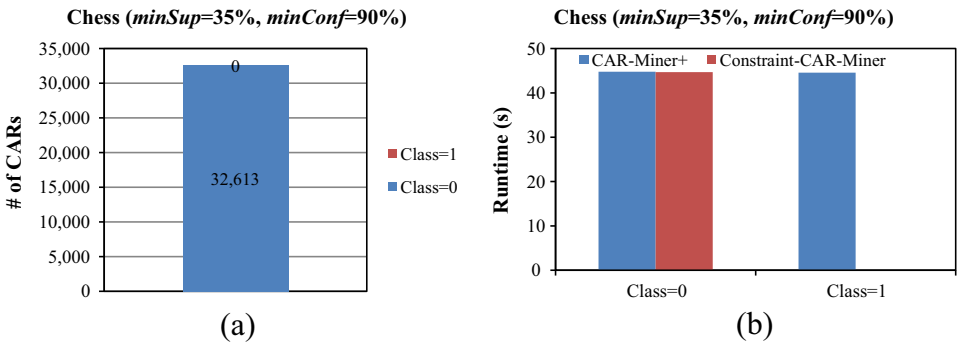

Fig. 14. (a) Number of constrained CARs for each class and (b) runtimes of CAR-Miner+ and Constraint-CAR-Miner for Chess dataset.

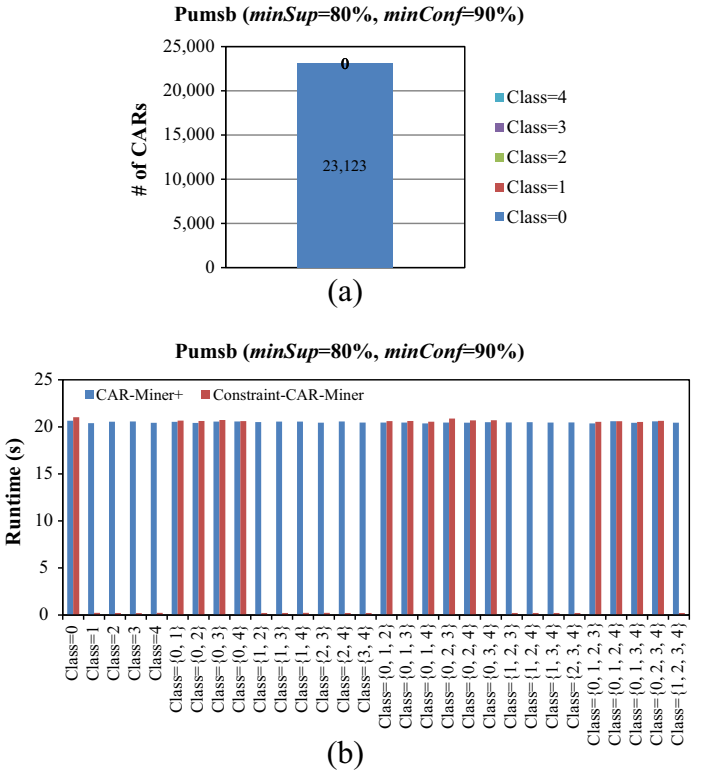

Fig. 15. (a) Number of constrained CARs for each class and (b) runtimes of CAR-Miner+ and Constraint-CAR-Miner for Pumsb dataset.

respectively. Similarly, the total proportion of four classes (1, 2, 3, and 4) is only around 5%, but that for Class 0 is nearly 95% in Pumsb.

5.2. Runtime vs Constraint_Class[1]

Experiments were conducted to determine the efficiency of the proposed algorithm. Figs. 9–16 show the execution times of CAR-Miner+ and Constraint-CAR-Miner with the class constraint for each dataset in Table 2. *minConf* = 90% was used for all experiments to obtain strong rules.

To initialize *Constraint_Class*, we define it as the power set of the set of class labels except the empty set and the whole set (i.e., there is no constraint). In other words, there are various $2^k - 2$ subsets in *Constraint_Class* (k is the number of classes).

[1] Note that in the experiments, we use minSup and minConf as percentages for concise representation.

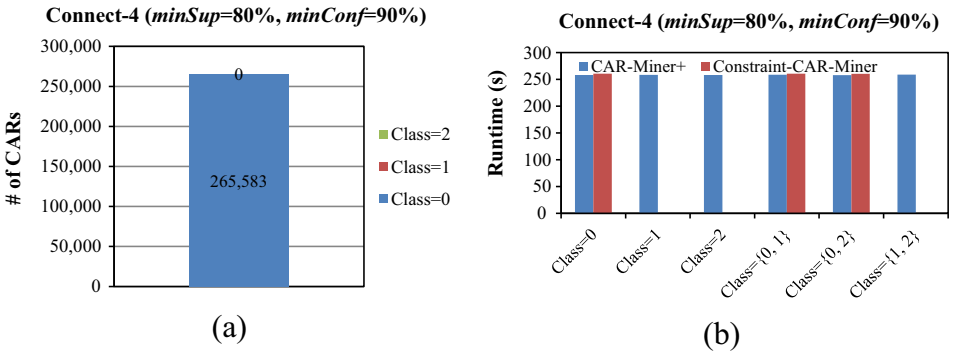

Fig. 16. (a) Number of constrained CARs for each class and (b) runtimes of CAR-Miner+ and Constraint-CAR-Miner for Connect-4 dataset.

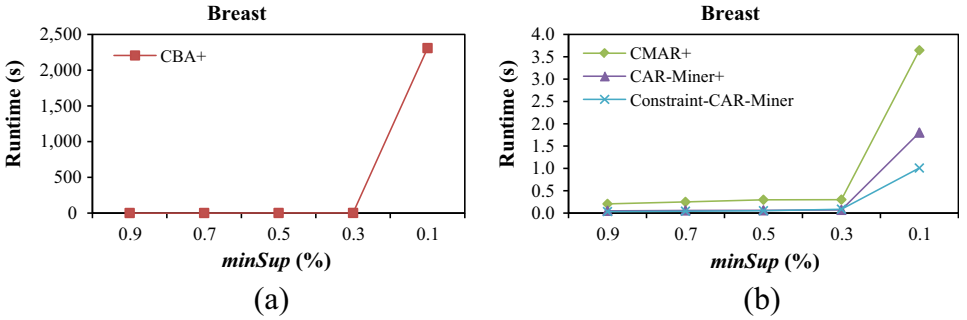

Fig. 17. Runtimes of (a) CBA+ and (b) CMAR+, CAR-Miner+, and Constraint-CAR-Miner for Breast dataset.

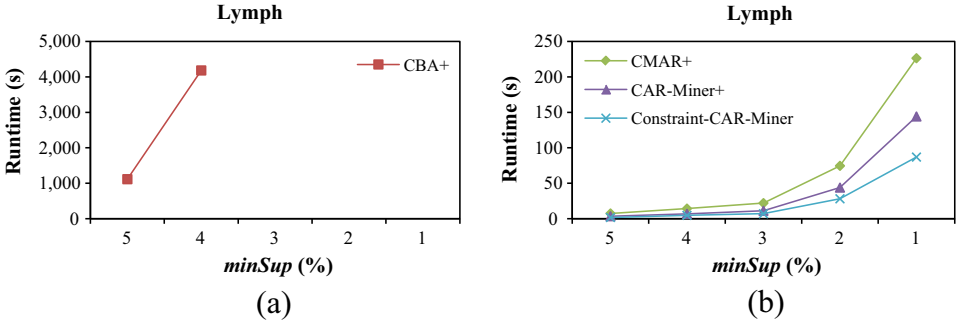

Fig. 18. Runtimes of (a) CBA+ and (b) CMAR+, CAR-Miner+, and Constraint-CAR-Miner for Lymph dataset.

For example, German has two classes (Classes 0 and 1). *Constraint_Class* has two subsets, $\{0\}$ and $\{1\}$. Similarly, Lymph has four classes (Classes 1, 2, 3, and 4). *Constraint_Class* has 14 cases: $\{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{1,2,3\}, \{1,2,4\}, \{1,3,4\}$, and $\{2,3,4\}$. Note that *Constraint_Class*=$\{1,2,3\}$ means finding rules that belong to Class 1, 2, or 3. In experiments, the algorithms were tested with each subset in *Constraint_Class*.

The results from Figs. 9–16 show that Constraint-CAR-Miner is better than CAR-Miner+ in most experiments. For example, consider the Breast dataset with *minSup* = 0.1% (Fig. 9(b)). The mining times of CAR-Miner+ are 0.890 s for Class 0 and 0.852 s for Class 1 while those of Constraint-CAR-Miner are only 0.519 s and 0.394 s, respectively. The speed ups are 42% and 64%, respectively. Similarly, consider the Lymph dataset with *minSup* = 1% (Fig. 10(b)). The mining times of CAR-Miner+ are 6.317 s, 6.315 s, 6.593 s, and 6.332 s for Classes 1, 2, 3, and 4, respectively, compared to 0.065 s, 4.434 s, 2.902 s, and 0.027 s of Constraint-CAR-Miner, respectively. The speed ups are 99%, 30%, 56%, and 99%, respectively. However, in some special cases where the number of constrained rules is approximately the total number of rules, then the execution time of Constraint-CAR-Miner is closer to (or higher than) CAR-Miner+ because it has to check more conditions than does CAR-Miner+. For example, consider Lymph with *minSup* = 1% and *Constraint_Class* $= \{2,3\}$. The mining time of Constraint-CAR-Miner is 7.417 s while that of CAR-Miner+ is 6.307 s. This can be explained by the domination of the number

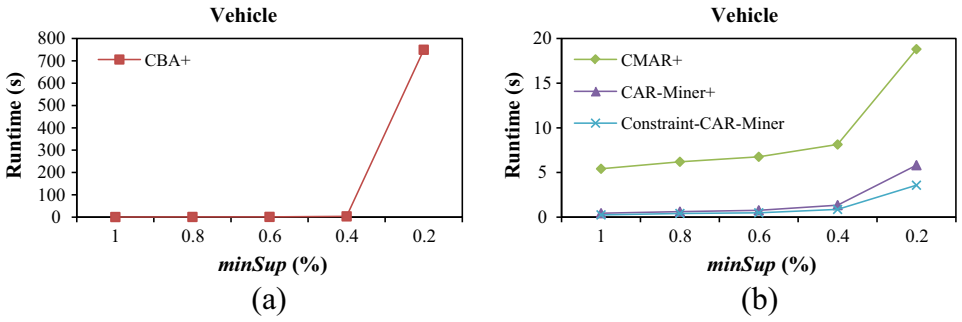

Fig. 19. Runtimes of (a) CBA+ and (b) CMAR+, CAR-Miner+, and Constraint-CAR-Miner for Vehicle dataset.

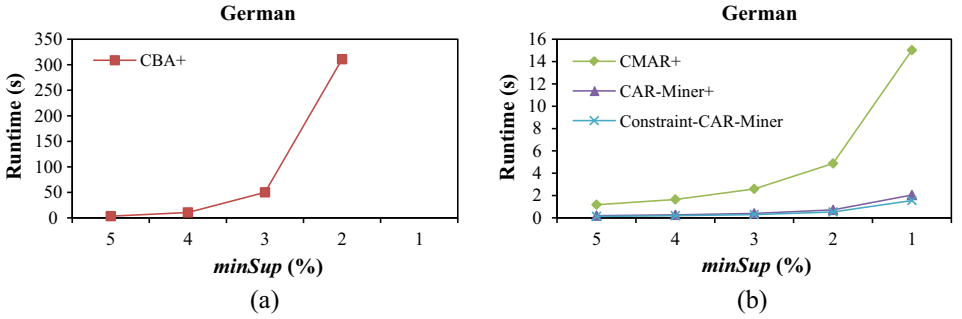

Fig. 20. Runtimes of (a) CBA+ and (b) CMAR+, CAR-Miner+, and Constraint-CAR-Miner for German dataset.

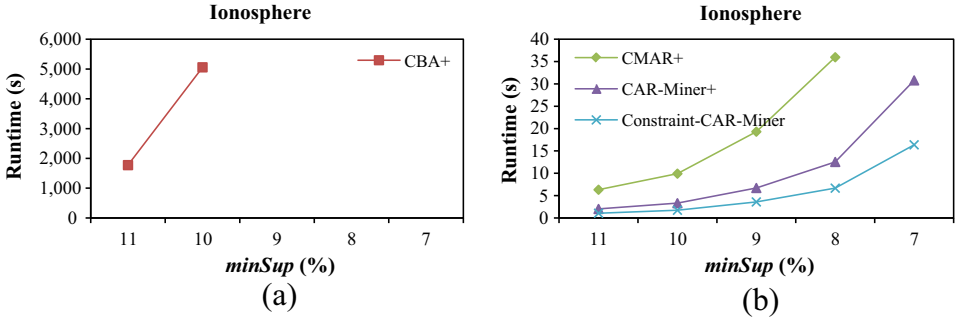

Fig. 21. Runtimes of (a) CBA+ and (b) CMAR+, CAR-Miner+, and Constraint-CAR-Miner for Ionosphere dataset.

of constrained rules belonging to Class 2 or 3; they make up 99% of the total number of rules, as shown in Fig. 10(a). In other words, the performance gap between the two algorithms is widened significantly when they are tested with a class whose number of constrained CARs is much less than the others. For example, Chess has 32,613 rules that belong to Class 0 whereas there are no rules in Class 1 with *minSup* = 35% (Fig. 14(a)). As a result, CAR-Miner+ takes 44.555 s to find rules in Class 1 while Constraint-CAR-Miner needs only 0.002 s to obtain the same result (speed up of 99.99%).

5.3. Runtime vs minSup[2]

In this section, Constraint-CAR-Miner is compared with CAR-Miner+, CBA+, and CMAR+ algorithms under various *minSup* values. Note that *minConf* = 90% was also used for all the experiments. Particularly, CBA+ is a modified version of CBA [23] for mining constrained CARs. CBA+ uses an Apriori-based method to generate candidates; it, however, counts the support of itemsets based on the intersection among transaction identifiers (*TIDs*) instead of scanning the dataset multiple times.

[2] Executable files of Constraint-CAR-Miner and other algorithms and experimental datasets can be downloaded from http://goo.gl/MB1OUe.

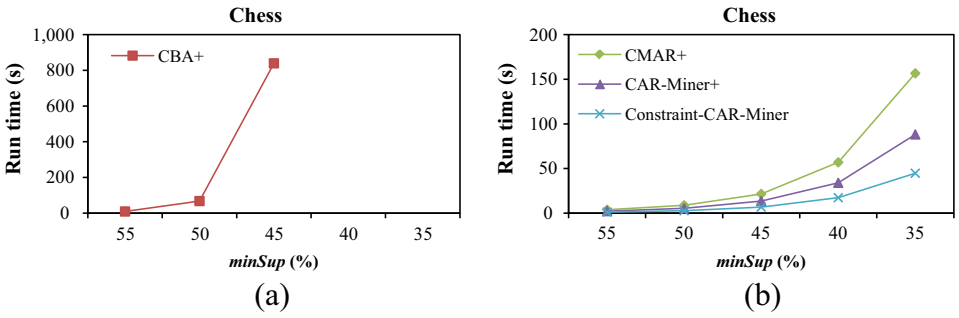

Fig. 22. Runtimes of (a) CBA+ and (b) CMAR+, CAR-Miner+, and Constraint-CAR-Miner for Chess dataset.

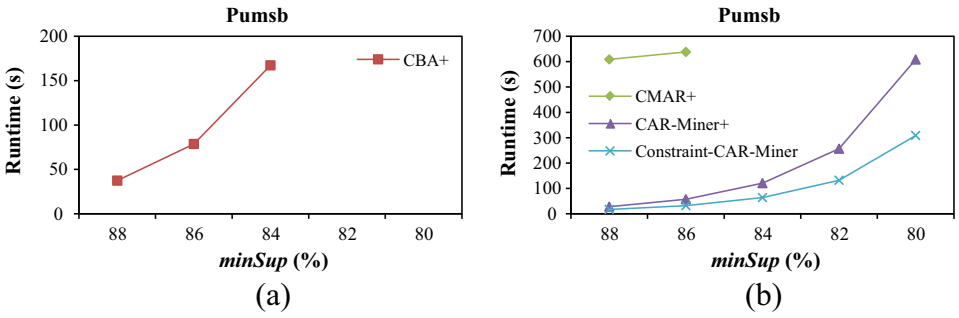

Fig. 23. Runtimes of (a) CBA+ and (b) CMAR+, CAR-Miner+, and Constraint-CAR-Miner for Pumsb dataset.

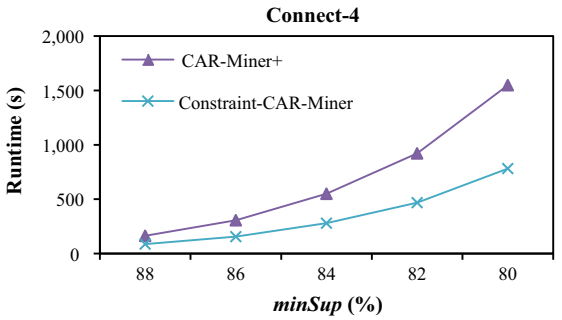

Fig. 24. Runtimes of CAR-Miner+ and Constraint-CAR-Miner for Connect-4 dataset.

This idea is also applied in MAC [1] and PAM [10]. Similarly, CMAR+ is an extension of CMAR [21] for mining rules with class constraints. Generally, CBA+ and CMAR+ are post-processing approaches. The results are shown in Figs. 17–24. Note that the runtime in the figures was calculated as the total runtime of testing with each case in *Constraint_Class*. For example, consider the Breast dataset with *minSup* = 0.1% and *minConf* = 90%. Constraint-CAR-Miner takes 0.519 s and 0.394 s to mine constrained CARs in Classes 0 and 1. This means that the runtime of Constraint-CAR-Miner is 0.913 s.

The results show that Constraint-CAR-Miner is much more efficient than the other algorithms in terms of mining time. For example, consider dataset Vehicle with *minSup* = 0.2% (Fig. 19. The runtime of Constraint-CAR-Miner is only 3.571 s compared to 749.755 s, 18.811 s, and 5.817 s of CBA+, CMAR+, and CAR-Miner+, respectively. For this instance, Constraint-CAR-Miner is 210 times faster than CBA+, 5.3 times faster than CMAR+, and 1.6 times faster than CAR-Miner+.

CBA+ works only on small datasets with high *minSup*. For instance, CBA+ fails to run on Lymph with *minSup* = 1–3% (Fig. 18(a)). CBA+ also cannot finish its work on large datasets such as Connect-4 (Fig. 24). CBA+ uses an Apriori-based method to generate candidates. When the value of *minSup* is very low, it is costly to handle a high number of candidates. CMAR+, CAR-Miner+, and Constraint-CAR-Miner have similar performance on small datasets (e.g., Breast, Lymph, Vehicle, and German). Nevertheless, the performance gap among these algorithms is widened significantly when they are tested on large datasets such as Chess, Pumsb, and Connect-4. Particularly, Constraint-CAR-Miner takes only 781.900 s to mine

Table 3
Compression ratio of Constraint-CAR-Miner over CAR-Miner+.

Dataset	*Constraint_Class*	# of generated nodes		Compression ratio (%)
		Constraint-CAR-Miner	CAR-Miner+	
Breast (*minSup* = 0.1%)	Class = 0	275,746	496,808	44
	Class = 1	221,062	496,808	56
Lymph (*minSup* = 1%)	Class = 1	3,648	4,040,347	99
	Class = 2	2,510,773	4,040,347	38
	Class = 3	1,508,060	4,040,347	63
	Class = 4	17,866	4,040,347	99
Vehicle (*minSup* = 0.2%)	Class = 1	16,255	127,400	87
	Class = 2	14,433	127,400	89
	Class = 3	40,613	127,400	68
	Class = 4	56,099	127,400	56
German (*minSup* = 1%)	Class = 0	573,915	608,759	6
	Class = 1	34,844	608,759	94
Ionosphere (*minSup* = 7%)	Class = 0	6,244,927	6,245,593	1
	Class = 1	666	6,245,593	99
Chess (*minSup* = 35%)	Class = 0	1,603,273	1,603,273	0
	Class = 1	0	1,603,273	100
Pumsb (*minSup* = 80%)	Class = 0	23,034	23,034	0
	Class = 1	0	23,034	100
	Class = 2	0	23,034	100
	Class = 3	0	23,034	100
	Class = 4	0	23,034	100
Connect-4 (*minSup* = 80%)	Class = 0	265,583	265,583	0
	Class = 1	0	265,583	100
	Class = 2	0	265,583	100

constrained rules on Connect-4 with *minSup* = 80% while CAR-Miner+ takes 1549.223 s to run and CMAR+ fails to run with the same parameters (Fig. 24). It can be concluded that the proposed algorithm, Constraint-CAR-Miner, is the fastest algorithm.

5.4. Compression ratio

The performance of Constraint-CAR-Miner is substantially improved by not having to generate unnecessary nodes (i.e., those that cannot form constrained rules). In other words, Constraint-CAR-Miner prunes more redundant nodes than does CAR-Miner+. This advantage minimizes the space storage required for the tree structure in Constraint-CAR-Miner, which reduces memory usage and speed ups execution time. Table 3 shows the compression ratio of Constraint-CAR-Miner over CAR-Miner+. Note that the compression ratio is defined as the proportion of the number of nodes pruned by Constraint-CAR-Miner to the number of nodes generated by CAR-Miner+.

Table 3 shows that the proposed method is capable of pruning a great number of redundant nodes. The compression ratio achieved by Constraint-CAR-Miner is considerable in most cases. The best case is for datasets Chess, Pumsb, and Connect-4, for which the compression ratio is 100%. For example, consider the Chess dataset with *minSup* = 35% and *Constraint_Class*={1}. Our algorithm does not generate any nodes on the tree whereas CAR-Miner+ generates more than one million nodes.

6. Conclusion and ideas for future work

This study proposed an efficient algorithm for mining CARs with class constraints based on a novel tree structure (NECR-tree). The proposed algorithm has three advantages. First, the algorithm can compute the support of itemsets quickly by using the list of $Obidset_i$. Second, it can directly identify the position *pos* of some nodes without computing the support. The confidence of a candidate rule is determined based on this information. Third, it does not need to generate nodes that cannot yield constrained rules based on some theorems, lemmas, and corollaries. With these improvements, an efficient and fast algorithm called Constraint-CAR-Miner for mining CARs with class constraints was presented. The experiments show that the proposed method outperforms the naive method and some existing methods.

However, a weakness of Constraint-CAR-Miner is that when the number of constrained rules in a given class dominates the total constrained rules; its performance becomes slower than the naive method. In the future, we will study how to minimize the dependence of the proposed method on the proportion of each class. We will also try to expand the algorithm to find CARs with constraints not only on the rule consequent but also on the rule antecedent. Mining such strictly constrained

rules may produce more interesting and useful rules from the perspective of the end user. Methods for mining frequent itemsets from incremental datasets have been developed recently [16,17,40]. Such methods can reduce execution time and memory usage compared to those obtained when rescanning the dataset. Thus, we will also try to integrate the techniques of incremental itemset mining into constrained CAR mining.

Acknowledgements

This work was funded by Vietnam's National Foundation for Science and Technology Development (NAFOSTED).

References

[1] Neda Abdelhamid, Aladdin Ayesh, Fadi Thabtah, Samad Ahmadi, Wael Hadi, MAC: a multiclass associative classification algorithm, J. Inf. Knowl. Manage. 11 (02) (2012) 1–10.
[2] Neda Abdelhamid, Fadi Thabtah, Aladdin Ayesh, Associative classification approaches: review and comparison, J. Inf. Knowl. Manage. 13 (3) (2014) 1–30.
[3] Rakesh Agrawal, Tomasz Imieliński, Arun Swami, Mining association rules between sets of items in large databases, in: Paper Presented at the ACM SIGMOD Record, 1993..
[4] Rakesh Agrawal, Ramakrishnan Srikant, Fast algorithms for mining association rules in large databases, in: Paper Presented at the 20th International Conference on Very Large Data Bases, 1994..
[5] Elena Baralis, Luca Cagliero, Tania Cerquitelli, Paolo Garza, Generalized association rule mining with constraints, Inf. Sci. 194 (2012) 68–84.
[6] Luca Cagliero, Paolo Garza, Improving classification models with taxonomy information, Data Knowl. Eng. 86 (2013) 85–101.
[7] Luca Cagliero, Paolo Garza, Itemset generalization with cardinality-based constraints, Inf. Sci. 224 (2013) 161–174.
[8] Ruichu Cai, Anthony Tung, Zhenjie Zhang, Zhifeng Hao, What is unequal among the equals? Ranking equivalent rules from gene expression data, IEEE Trans. Knowl. Data Eng. 23 (11) (2011) 1735–1747.
[9] Chun-Hao Chen, Rui-Dong Chiang, Cho-Ming Lee, Chih-Yang Chen, Improving the performance of association classifiers by rule prioritization, Knowl.-Based Syst. 36 (2012) 59–67.
[10] Fuzan Chen, Yanlan Wang, Minqiang Li, Harris Wu, Jin Tian, Principal association mining: an efficient classification approach, Knowl.-Based Syst. 67 (2014) 16–25.
[11] Houtao Deng, George Runger, Eugene Tuv, Wade Bannister, CBC: an associative classifier with a small number of rules, Dec. Supp. Syst. 59 (2014) 163–170.
[12] Jie Dong, Min Han, BitTableFI: an efficient mining frequent itemsets algorithm, Knowl.-Based Syst. 20 (4) (2007) 329–335.
[13] Hai Duong, Tin Truong, Bac Le, An Efficient Algorithm for Mining Frequent Itemsets with Single Constraint Advanced Computational Methods for Knowledge Engineering, Springer, 2013. pp. 367–378.
[14] Jiawei Han, Jian Pei, Yiwen Yin, Mining frequent patterns without candidate generation, in: Paper Presented at the ACM SIGMOD Record, 2000..
[15] Raudel Hernández-León, José Hernández-Palancar, J.A. Carrasco-Ochoa, José Fco Martínez-Trinidad, Studying Netconf in Hybrid Rule Ordering Strategies for Associative Classification Pattern Recognition, Springer, 2014. pp. 51–60.
[16] Tzung-Pei Hong, Ching-Yao Wang, An efficient and effective association-rule maintenance algorithm for record modification, Exp. Syst. Appl. 37 (1) (2010) 618–626.
[17] Tzung-Pei Hong, Ching-Yao Wang, Shian-Shyong Tseng, An incremental mining algorithm for maintaining sequential patterns using pre-large sequences, Exp. Syst. Appl. 38 (6) (2011) 7051–7058.
[18] Carson Kai-Sang Leung, Laks V.S. Lakshmanan, Raymond T Ng, Exploiting succinct constraints using FP-trees, ACM SIGKDD Explor. Newslett. 4 (1) (2002) 40–49.
[19] Jiye Li, Nick Cercone, Discovering and ranking important rules, in: Paper presented at the IEEE International Conference on Granular Computing, 2005..
[20] Wei Li, Longbing Cao, Dazhe Zhao, Xia Cui, Jinzhu Yang, CRNN: integrating classification rules into neural network, in: Paper Presented at the International Joint Conference on Neural Networks (IJCNN 2013), 2013..
[21] Wenmin Li, Jiawei Han, Jian Pei, CMAR: accurate and efficient classification based on multiple class-association rules, in: Paper Presented at the IEEE International Conference on Data Mining (ICDM 2001), 2001..
[22] Wen-Yang Lin, Ko-Wei Huang, Chin-Ang Wu, MCFPTree: an FP-tree-based algorithm for multi-constraint patterns discovery, Int. J. Bus. Intell. Data Min. 5 (3) (2010) 231–246.
[23] Bing Liu, Wynne Hsu, Yiming Ma, Integrating classification and association rule mining, in: Paper presented at the 4th International Conference on Knowledge Discovery and Data Mining (KDD 1998), 1998..
[24] Huawen Liu, Lei Liu, Huijie Zhang, A fast pruning redundant rule method using Galois connection, Appl. Soft Comput. 11 (1) (2011) 130–137.
[25] Nan Lu, Chun-Guang Zhou, Jing-Zhou Zhou, Research on association rules mining algorithm with item constraints, in: Paper Presented at the International Conference on Cyberworlds, 2005..
[26] Moath Najeeb, A.E. Sheikh, Mohammed Nababteh, A new rule ranking model for associative classification using a hybrid artificial intelligence technique, in: Paper presented at the 3rd International Conference on Communication Software and Networks (ICCSN 2011), 2011..
[27] Raymond T. Ng, Laks V.S. Lakshmanan, Jiawei Han, Alex Pang, Exploratory mining and pruning optimizations of constrained associations rules, in: Paper Presented at the ACM SIGMOD International Conference on Management of Data, 1998..
[28] Dang Nguyen, Bay Vo, Bac Le, Efficient strategies for parallel mining class association rules, Exp. Syst. Appl. 41 (10) (2014) 4716–4729.
[29] Loan T.T. Nguyen, Bay Vo, Tzung-Pei Hong, Hoang Chi Thanh, Classification based on association rules: a lattice-based approach, Exp. Syst. Appl. 39 (13) (2012) 11357–11366.
[30] Loan T.T. Nguyen, Bay Vo, Tzung-Pei Hong, Hoang Chi Thanh, CAR-Miner: an efficient algorithm for mining class-association rules, Exp. Syst. Appl. 40 (6) (2013) 2305–2311.
[31] Jian Pei, Jiawei Han, Laks V.S. Lakshmanan, Mining frequent itemsets with convertible constraints, in: Paper presented at the 17th International Conference on Data Engineering, 2001..
[32] Wei Song, Bingru Yang, Zhangyan Xu, Index-BitTableFI: an improved algorithm for mining frequent itemsets, Knowl.-Based Syst. 21 (6) (2008) 507–513.
[33] Ramakrishnan Srikant, Quoc Vu, Rakesh Agrawal, Mining association rules with item constraints, in: Paper presented at the 3rd International Conference on Knowledge Discovery and Data Mining (KDD 1997), 1997..
[34] Fadi Thabtah, Peter Cowling, Yonghong Peng, MMAC: a new multi-class, multi-label associative classification approach, in: Paper presented at the 4th IEEE International Conference on Data Mining (ICDM 2004), 2004..
[35] Risi Thonangi, Vikram Pudi, ACME: an associative classifier based on maximum entropy principle, in: Paper Presented at the 16th International Conference on Algorithmic Learning Theory, 2005..
[36] Hannu Toivonen, Mika Klemettinen, Pirjo Ronkainen, Kimmo, Hätönen, Heikki Mannila, Pruning and grouping discovered association rules, in: Paper Presented at the ECML-95 Workshop on SMLKD, 1995..

[37] Anh Tran, Hai Duong, Tin Truong, Bac Le, Efficient algorithms for mining frequent itemsets with constraint, in: Paper presented at the 3rd International Conference on Knowledge and Systems Engineering (KSE 2011), 2011..
[38] Bay Vo, Tzung-Pei Hong, Bac Le, DBV-Miner: a dynamic bit-vector approach for fast mining frequent closed itemsets, Exp. Syst. Appl. 39 (8) (2012) 7196–7206.
[39] Bay Vo, Bac Le, A Novel Classification Algorithm Based on Association Rules Mining Knowledge Acquisition: Approaches, Algorithms and Applications, vol. 5465, Springer, 2009.
[40] Bay Vo, Tuong Le, Tzung-Pei Hong, Bac Le, An effective approach for maintenance of pre-large-based frequent-itemset lattice in incremental mining, Appl. Intell. 41 (3) (2014) 759–775.
[41] Xiaoxin Yin, Jiawei Han, CPAR: classification based on predictive association rules, in: Paper Presented at the 3rd SIAM International Conference on Data Mining (SDM 2003), 2003..
[42] Mohammed Zaki, C.-J. Hsiao, Efficient algorithms for mining closed itemsets and their lattice structure, IEEE Trans. Knowl. Data Eng. 17 (4) (2005) 462–478.
[43] Mohammed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, New algorithms for fast discovery of association rules, in: Paper presented at the 3rd International Conference on Knowledge Discovery and Data Mining, 1997..

View publication stats