An Efficient Construction and Application Usefulness of Rectangle Greedy Covers[☆]

Koji Ouchi, Atsuyoshi Nakamura, Mineichi Kudo

Graduate School of Information Science and Technology, Hokkaido University
Kita 14, Nishi 9, Kita-ku, Sapporo, Hokkaido, 060-0814, Japan
{*oouchi, atsu, mine*}*@main.ist.hokudai.ac.jp*

Abstract

We develop efficient construction methods of a rectangle greedy cover (RGC), and evaluate its usefulness in applications. An RGC is a greedy cover of the set of given positive instances by *exclusive* axis-parallel hyperrectangles, namely, axis-parallel hyperrectangles that exclude all the given negative instances. An RGC is expected to be a compact classification rule with high readability because the number of its component rectangles is expected to be small and it can be seen as a disjunctive normal form, which is one of the most readable representations for us.

We propose two approaches of RGC construction: enumeration approach and direct approach. In enumeration approach, the maximal exclusive positive subsets (MEPSs) are enumerated first and then an ordinary greedy set covering is done using the enumerated MEPSs. We make clear the relation between enumeration of the maximal frequent itemsets and enumeration of the MEPSs, and convert an efficient enumeration algorithm LCMmax [1] of maximal frequent itemsets to an enumeration algorithm LCMmax.R_{naive} of MEPSs. We also develop a more efficient version of LCMmax.R_{naive}, or LCMmax.R, by incorporating effective dynamic reordering of instances using excluded frequency and bit-parallel exclusiveness check. In direct approach, each component MEPS of an RGC is searched not from enumerated MEPSs but directly from the dataset that consists of the remaining uncovered positive instances and the whole negative instances. We developed an algorithm called MRF that efficiently finds an maximum-sized MEPS for given positive and negative instances. MRF is made from LCMmax.R by modifying it so as to find a maximum-sized MEPS only. An RGC is constructed by MRF repetition, that is, by repeatedly executing MRF using the remaining uncovered positive instances.

According to our experimental evaluation using UCI-repository datasets, LCMmax.R was about 5-11 times faster than LCMmax.R_{naive}, which indicates effectiveness of the introduced two improvements. MRF repetition, however, was significantly faster than LCMmax.R, and it was enough fast to use practically for small datasets. The experimental results using UCI-repository datasets also showed that accuracy of a nearest rectangle classifier using an RGC is close to that using the hyperrectangles output by the randomized subclass method (RSM) [2] though the number of component rectangles of an RGC is significantly smaller than the number of the hyperrectangles output by RSM. The performance of RGC was also shown to be comparable to that of the six popular classifiers including logistic regression and support vector machine. The disjunctive normal form representation of the classification rules obtained by RGC was demonstrated to be simpler and more readable for us than that obtained by RSM and C4.5.

Keywords: greedy cover, axis-parallel hyperrectangle, classification, data mining

1. Introduction

In a finite dimensional Euclidean space, an axis-parallel hyperrectangle is one of the most usable shapes as a component of a region. The region of an axis-parallel hyperrectangle in d-dimensional space is compactly represented by a conjunction $\bigwedge_{i=1}^{d}(a_i \le x_i \le b_i)$ using $2d$ boundary values $a_i, b_i (i = 1, 2, ..., d)$, which is highly readable for human

[☆]A preliminary version was presented in GrC 2011 [3].

being. (Consider the case that attributes x_i are height, weight, age, etc.) An approximation of a more complex region can be obtained using a union of axis-parallel hyperrectangles, which is represented by a disjunction of conjunctions, namely, a DNF(Disjunctive Normal Form). A DNF is known to be one of the most readable rule representations for human being.

In the area of pattern recognition and machine learning, a union of axis-parallel hyperrectangles has been used as a generalized compressed representation for all the given instances of each class. In the representation, a group X of clustered instances of the same class is compactly represented by the smallest axis-parallel hyperrectangle $r(X)$ that contains X. The rectangle $r(X)$, which is unique and easily calculated, is used instead of X in a nearest neighbour classifier for space- and time-efficiency [2, 4, 5, 6]. In data mining, mining unions of axis-parallel hyperrectangles was studied to obtain human-readable useful rules [7].

According to Occam's razor [8], the simplest hypothesis is the best one among those with similar accuracy for a training data. In that meaning, among unions of rectangles that are consistent with a given dataset, the union with the minimum number of rectangles should be selected as a generalized representation of a given dataset. The problem of finding such a union of rectangles, however, can be seen as an instance of an NP-hard problem called the *minimal set cover problem*, because a consistent union of rectangles for each class is a cover of the instance set of the class by *exclusive* rectangles, namely, the rectangles that include no instance of the other classes. As an approximate solution of the minimal set cover problem, a greedy cover is popular because it is easily calculated and its size is close to the minimum in many practical data. For given positive and negative instance sets, S^+ and S^-, respectively, a greedy cover of S^+ by exclusive rectangles is called a *rectangle greedy cover (RGC)* through this paper.

In this paper, we study efficient construction algorithms of an RGC and its usefulness in classification and rule mining. An RGC is known to be constructed in polynomial time when the dimensions of an instance space is fixed [8]. RGCs, however, have not been used practically because the known algorithms are too slow for practical use. To our best knowledge, this is the first research in which RGCs are constructed and their application performance is measured for real data.

An RGC is a cover of a given set S^+ of positive instances by exclusive rectangles, and the RGC construction can be seen as an ordinary greedy set covering of S^+ by *exclusive positive subsets (EPSs)*, which are subsets of S^+ covered by some exclusive rectangle. Different from an ordinary set covering problem, the family of EPSs is given not explicitly, but implicitly through a set S^- of negative instances. Considering this difference, there are two approaches of the RGC construction: enumeration and direct approaches. In enumeration approach, maximal EPSs (MEPSs) are enumerated first and then S^+ is greedily covered by some of the enumerated MEPSs, while in direct approach, it is repeated to directly find an MEPS that covers the remaining positive instances most using S^- every time.

As for enumeration approach, fortunately, enumeration of the MEPSs is known as an instance of a more general problem [9, 10] which includes enumeration of the maximal frequent itemsets (MFIs), for which efficient algorithms have been actively developed recently. We make clear the relation between enumeration of the MEPSs and that of the MFIs, and show how to convert an algorithm for the MFIs to that for the MEPSs. According to the way of conversion, we developed an algorithm called LCMmax.$\mathrm{R}_{\mathrm{naive}}$ from LCMmax [1], which is one of the fastest algorithms in enumeration of the MFIs. Furthermore, we developed a more efficient version of LCMmax.$\mathrm{R}_{\mathrm{naive}}$ called LCMmax.R by incorporating two improvements: effective dynamic reordering of instances using excluded frequency and bit-parallel exclusiveness check.

As for direct approach, we developed an algorithm called MRF repetition, which repeatedly executes MRF(Maximal Rectangle Finder) for the remaining uncovered positive instances. Here, MRF is an algorithm that is made from LCMmax.R by modifying it so as to find a maximum-sized MEPS only. MRF is faster than LCMmax.R by the effect of pruning by the maximum size found so far and removal of maximality check. However, it is not trivial whether MRF repetition is faster than LCMmax.R with an ordinary greedy set covering process because MRF must be executed multiple times in the former algorithm while LCMmax.R only has to be executed once in the latter one.

We conducted experiments to demonstrate computational efficiency of our algorithms and the usefulness of an RGC using 13 real datasets of UCI repository [11]. We observed the effect of our two improvements that are incorporated into LCMmax.R: LCMmax.R was about 5-11 times faster than LCMmax.$\mathrm{R}_{\mathrm{naive}}$ for the six datasets we used in this experiment. MRF repetition, however, was faster than LCMmax.R for all the datasets but the haberman dataset. Especially for the datasets with a large number of MEPSs, MRF repetition ran significantly faster than LCMmax.R: 127 time faster for vehicle dataset. MRF repetition constructed an RGC within 10 seconds for 10 datasets and within 15 minutes for all the datasets including the rest 3 datasets, which indicated practical usability for small datasets.

As a classifier by the class label of the nearest rectangle, the classification performance of an RGC was shown to be close to that of the rectangles output by the randomized subclass method (RSM) [2], though the RGCs consist of a significantly smaller number of rectangles than the sets of rectangles output by RSM, where RSM is a conventional construction method of an exclusive-rectangle set by repeated random generation of an MEPS. In our experiments, the classification performance of an RGC was also shown to be comparable to that of popular 6 classifiers: AdaBoost, logistic regression with a ridge estimator, Naive Bayes, SVM, C4.5(J48) and k-nearest neighbor. Compared with RSM and C4.5 [12], consistent DNF-rules produced by RGCs were demonstrated to be simpler unless DNF-rules produced by C4.5 had accuracy less than 0.9.

This paper is organized as follows. In Sec. 2, we briefly review previous work related to RGCs. RGCs are defined, their applications are shown, and their construction hardness is explained in Sec. 3. In Sec. 4, how to convert MFI-enumeration algorithm to MEPS-enumeration algorithm is explained first, then a naive and a more efficient version of the algorithms, LCMmax.$\mathrm{R}_{\mathrm{naive}}$ and LCMmax.R, respectively, which are converted from LCMmax, are shown. MRF is also described as a modification of LCMmax.R in the section. The experimental results on efficiency of the proposed algorithms and application usefulness of RGCs are reported in Sec. 5. Some concluding remarks are described in Sec. 6.

2. Related Work

In the area of computational learning theory, it is studied whether the region represented by a union of axis-parallel hyperrectangles in d-dimensional Euclidean space is learnable from examples. Blumer et al.[8] proved polynomial PAC(Probably Approximately Correct)-learnability of the class of the regions that is represented by a union of axis-parallel hyperrectangles in d-dimensional Euclidean space for a fixed constant d. However, the algorithm used in the proof is too slow for practical use.

In the area of pattern recognition and machine learning, there is a study in which a rectangle is used instead of a group of instances of the same class in order to make a nearest neighbor classifier more space- and time-efficient. The nearest neighbor classifier using rectangles classifies an instance x into the class of the rectangle that is the nearest to x. There are variations on what sets of rectangles are used for a representation of a whole dataset.

Kudo et al. proposed an approach in which the set of rectangles are composed of the minimum rectangles $r(X)$ containing MEPSs X. They proposed a method that uses all the MEPSs [4], but the proposed MEPS enumeration algorithm was too slow to use practically. Later, they proposed the randomized subclass method (RSM) [2], which uses k randomly generated MEPSs for each class, where k is an input parameter. An RGC needs more time to construct than a set of rectangles constructed by RSM but uses a smaller number of rectangles in most cases without degrading classification performance so much.

As for methods using exclusive subsets of the same class, there is a method called BNGE (Batch NGE) [6]. In BNGE, a set of rectangles is generated by regarding each instance itself as a rectangle first and then repeatedly merging two rectangles X and Y of the same class to the minimal rectangle $r(X \cup Y)$ containing $X \cup Y$ unless $r(X \cup Y)$ overlaps some rectangles of the other classes. Better performance was reported [6] for BNGE compared to NGE [5], which allows the use of non-exclusive subsets X of the same class, that is, $r(X)$ can contain instances of other classes as exceptions. BNGE is more unstable than RGCs because the set of rectangles used by BNGE varies depending on the order of merging.

Enumeration of the MEPSs was also studied in [10] as an instance of a more general problem [9] which includes frequent maximal itemset mining. The proposed enumeration algorithm in [10] is not fast though it also let us know the VC dimension of the intersection closure of the MEPS set.

A greedy set cover is an approximate solution of the minimal set cover problem, which is known to be NP-hard [13]; the number of covering components is guaranteed to be at most $k \ln m + 1$, where k is the number of components of a minimal cover and m is the size of the set to be covered. A greedy set cover is often used in the situation that a small-sized cover is desirable because it is obtainable in polynomial time and its size is much smaller than the above upper bound in most practical problems. Set covering machine (SCM) proposed by Marchand ans Shawe-Taylor [14] constructs a cover of the set of positive instances in a greedy manner. They evaluate a covering component h by *usefulness* value which is defined as $|Q_h| - p|R_h|$, where $|Q_h|$ and $|R_h|$ are the numbers of covered positive and negative instances, respectively, and p is a penalty parameter. By using MEPSs as covering components and setting p to ∞,

SCM outputs an RGC. However, the description of SCM algorithm provides no information on how to efficiently construct an RGC.

3. Rectangle Greedy Covers

3.1. Definitions

Let S^+ and S^- denote the finite sets of positive and negative instances in Euclidean space $\mathbb{R}^d$. Let $\mathcal{R}$ denote the set of all (hyper)rectangles[1] parallel to the axes in $\mathbb{R}^d$. A rectangle $R \in \mathcal{R}$ is said to be *exclusive* if R contains no negative instances, i.e., $R \cap S^- = \emptyset$. We also use the word 'exclusive' for a subset X of S^+; a set $X \subseteq S^+$ is *exclusive* if X is contained in some exclusive rectangle $R \in \mathcal{R}$. For any finite set $X \subset \mathbb{R}^d$, $r(X)$ denotes the minimal rectangle among the rectangles R that contain X, i.e.,

$$r(X) = \Pi_{i=1}^{d}[\min\{x_i : (x_1, x_2, ..., x_d) \in X\}, \max\{x_i : (x_1, x_2, ..., x_d) \in X\}].$$

For any $X \subseteq S^+$,

$$X \text{ is exclusive} \Leftrightarrow r(X) \text{ is exclusive}$$

holds by virtue of the minimality of $r(X)$. For a set $X \subseteq S^+$, a set $r(X) \cap S^+$ is said to be the *closure of* X, and X is called a *closed set* if the closure of X is equal to X. Let $\Omega(S^+, S^-, \mathcal{R})$ denote the family of the *maximal exclusive positive subsets (MEPSs)*, i.e.,

$$\begin{aligned}&\Omega(S^+, S^-, \mathcal{R})\\ &= \{X : X \subseteq S^+, r(X) \cap S^- = \emptyset, r(X \cup \{x\}) \cap S^- \neq \emptyset \text{ for all } x \in S^+\}.\end{aligned}$$

It is trivial from the definition that all the sets in $\Omega(S^+, S^-, \mathcal{R})$ are closed sets.

A *rectangle greedy cover (RGC)* $\{R_1, R_2, ..., R_k\} \subset \mathcal{R}$ is a greedily constructed cover of S^+ by exclusive rectangles, that is, $\bigcup_{i=1}^{k} R_i \cap S^+ = S^+$ and

$$\emptyset \neq R_i = \underset{R:R\in\mathcal{R},R\cap S^-=\emptyset}{\arg\max} \left| \left(R \setminus \bigcup_{j=1}^{i-1} R_j \right) \cap S^+ \right| \tag{1}$$

for $i = 1, 2, ..., k$, where $|\cdot|$ is the number of elements in a set '$\cdot$'. Note that there exists an MEPS X_i such that $R_i = r(X_i)$ satisfies Equation (1). This means that one RGC $\{r(X_1), r(X_2), ..., r(X_k)\}$ can be obtained from a greedy cover $\{X_1, X_2, ..., X_k\} \subseteq \Omega(S^+, S^-, \mathcal{R})$ of S^+.

3.2. RGC Classifiers

An RGC can be seen as a representation of positive regions, so using an RGC we can decide whether an arbitrary instance is positive or not depending on whether the instance is inside the RGC or not. In the case that the positive region belongs to the class of unions of axis-parallel hyperrectangles, an algorithm of constructing an RGC is known to be a polynomial-time PAC(Probably Approximately Correct)-learner if d is a fixed constant [8]. However, it is not practical to directly use an RGC for classification like the above because a union of (a small number of) axis-parallel hyperrectangles is too coarse as an approximation of positive regions in a real classification problem.

There is another usage of exclusive hyperrectangles in which a hyperrectangle is used not as a representation of positive region but as a generalized compressed representation of a cluster of positive instaces. By this usage of exclusive hyperrectangles, instances of each class might be compactly represented by a set of hyperrectangles even in multiclass case if we extend the definition of a *exclusive hyperrectangle* as a hyperrectangle that contains no instance of the other classes. In such compact representation of given instances, it is quite natural to consider a classifier using nearest neighbor criterion. In fact, Salzberg porposed such a classifier, called a *nearest rectangle classifier* here, in

[1] We use the word 'rectangle' also for a hyperrectangle in this paper.

which each class is represented by a set of hyperrectangles and an instance is classified into the class of the nearest rectangle, and good performance was reported on the nearest rectangle classifiers using certain sets of rectangles [5]. An RGC can be used as a representation of instances of each class in a nearest rectangle classifier. The computational time needed for classification by a nearest rectangle classifier linearly depends on the number of rectangles used in the classification. In the case that the number of rectangles is much smaller than the number of instances, the nearest rectangle classifier is significantly faster than the popular instance-based nearest neighbor method, whose computation time linearly depends on the number of instances. So by using RGCs, faster classification is expected due to a small number of rectangles in RGCs.

3.3. RGC rules

As studied in [7], each exclusive rectangle can be seen as a conjunction rule for the positive class. For example, an exclusive rectangle $R = \prod_{i=1}^{d}[a_i, b_i]$ represents a rule

$$\textbf{if } x_1 \in [a_1, b_1] \wedge x_2 \in [a_2, b_2] \wedge \cdots \wedge x_d \in [a_d, b_d]$$
$$\textbf{then } (x_1, x_2, \ldots, x_d) \text{ belongs to the positive class.}$$

Thus, an RGC can be seen as a DNF (Disjunction Normal Form) representation of a classification rule for the positive class. Note that DNF is known to be one of the most readable rule representations for human being. A small number of rectangles in an RGC results in a DNF with fewer component conjunctions, which improves readability further. Each conjunction in an obtained DNF can be further simplified by removing boundaries unnecessary for consistency to the given training sets S^+ and S^-. (See [7] in detail.)

3.4. Hardness of Constructing an RGC

In spite of the above merits of using RGCs in classification and rule mining, high computational cost of constructing an RGC has been preventing it from being practically used. The polynomial-time PAC-learning algorithm, which is used in the proof of a theorem in [8], first enumerates[2] the MEPSs and then does set covering using $r(X)$ for the enumerated sets X in a greedy manner. However, the computational cost for enumerating the MEPSs by naive algorithms is very high. For example, one of the naive algorithms is the following: for each subset X of S^+ with at most $2d$ instances[3], calculate $r(X) \cap S^+$ and check its exclusiveness and maximality. Such an algorithm needs $\Omega((\frac{m^+}{2d})^{2d} \times m^+ d \times m^- d) = \Omega((\frac{m^+}{2d})^{2d} m^+ m^- d^2)$, which is exponential to d, where $m^+ = |S^+|$ and $m^- = |S^-|$.

4. RGC Construction Algorithms

In this section, we study efficient algorithms that construct an RGC $\{r(X_1), r(X_2), ..., r(X_k)\}$ using MEPSs $X_1, X_2, ..., X_k$, i.e., $\{X_1, X_2, ..., X_k\} \subseteq \Omega(S^+, S^-, \mathcal{R})$. There are mainly two approaches to construct such an RGC.

One is an approach with two separated processes: enumerating the MEPSs and covering S^+ greedily using them. In this approach, the MEPS enumeration, which is called as subclass problem in [4], must be done efficiently. As an algorithm for this approach, we develop an algorithm called LCMmax.R for the MEPS enumeration, which is based on an efficient algorithm called LCMmax for enumerating maximal frequent itemsets (MFIs).

The other is a direct approach: repeatedly finding an MEPS that covers the largest number of remaining uncovered instances without enumerating the MEPSs. An algorithm for finding a maximum-sized MEPS, called MRF, can be obtained by modifying LCMmax.R so as to find only a maximum-sized one instead of all the maximal ones.

In the next subsection, we explain how to obtain an enumeration algorithm for the MEPSs from an enumeration algorithm for the MFIs. Then, our algorithms used in the above two approaches, (a naive and an improved version of) LCMmax.R and MRF, are described in the following three subsections.

[2] Their algorithm enumerates subsets $R \cap S^+$ for all $R \in \mathcal{R}$ but the MEPSs are enough for greedy covering of S^+.

[3] $2d$ instances are enough to generate all possibilities because the number of boundaries are $2d$.

4.1. Converting Algorithms from MFI Mining

It is very useful for marketing to know what combinations of items are frequently bought together. Enumerating frequent itemsets, i.e., itemsets that are frequently contained by transactions in a given transaction database, is a popular way to find such combinations of items in the area of data mining. As indicated in [10], the problem of enumerating the MEPSs has the same abstract structure as the problem of enumerating the MFIs in a given transaction database. In the followings, we clarify the correspondence of the two problems.

The essential similarity comes from the similarity between exclusiveness property for subsets X of S^+ and frequentness property for itemsets I in a transaction database $\mathcal{T}$ for a minimum support σ. Both are antimonotone, i.e., if a subset X (an itemset I) is not exclusive (frequent), neither any one of its supersets is. This means that an enumeration algorithm for exclusive subsets of S^+ can be obtained from that for frequent itemsets by replacing the frequentness checks with the exclusiveness checks. Furthermore, both properties are invariant to the closure operation, i.e., if a subset X (an itemset I) is exclusive (frequent), then its closure $r(X) \cap S^+(\bigcap_{T\in\mathcal{T}, I\subseteq T} T)$ is also exclusive (frequent). This fact enables us to obtain a faster algorithm for the problem restricted to *closed* sets from a faster algorithm for the corresponding problem in itemset mining. Invariantness to the closure operation guarantees that maximal sets with the property are closed sets. Thus, enumeration algorithms for maximal sets with the property can use the search space pruning available for closed-set enumeration.

Enumeration of frequent itemsets is considered as an important problem as mentioned above, and efficient algorithms for such enumeration have been actively studied. The algorithms for MFI mining which finishes computation within practical time has been already proposed. The corresponding algorithms for the MEPSs, which can be constructed by converting such fast algorithms for the MFIs using the correspondence described above, are expected to be also practically fast.

However, there are some points in which enumeration of the MEPSs seems more time-consuming than enumeration of the MFIs.

First, the exclusiveness check seems more heavy task than the frequentness check. The frequentness check of a super set I' of I can be done efficiently using the set of transactions $\mathcal{T}_I$ that contain I because the set of transactions that contain I' is a subset of $\mathcal{T}_I$. Unfortunately, we cannot find any good property that helps efficient calculation of the exclusiveness check. In the exclusiveness check of X, for all the negative instances x, it must be checked whether x is contained in $r(X)$ or not, and there is no trivial way to reduce the number of negative instances to check.

Second, MEPSs are generally larger than MFIs in real applications. In most practical cases, the positive space is composed of a few continuous regions, so there is a high possibility of existence of large rectangles that contain a lot of positive instances. By contrast, there are many MFI-mining applications in which only small MFIs to enumerate are there. For example, the number of items bought together is generally not so large. Furthermore, frequentness is determined by the minimum support parameter σ, and the MFI sizes become smaller by using larger σ. For the MEPS mining, there is no such parameter that can reduce the MEPS sizes.

Finally, the number of the MEPSs is generally larger than the number of the MFIs. For example, consider the case in which both positive and negative instances are generated randomly according to uniform distribution on $[0,1]^d$. Let X be the set of randomly selected two positive instances and let n be the number of negative instances. Then, the probability that $r(X)$ is exclusive is $(1-1/3^d)^n$, which is high even for small d if n is not huge: 0.662 for $d = 5, n = 100$, 0.998 for $d = 10, n = 100$ and 0.983 for $d = 10, n = 1000$. In real situation, the probability is higher than that because two instances in the same class tend to appear close to each other. Thus, the number of exclusive positive subsets is expected to be large, and so is the number of MEPSs. On the other hand, the probability that a set of two items is frequent seems not so high in many applications. For example, the number of two items that are frequently bought together seems much smaller than the number of two items that are rarely bought together.

The algorithms for enumerating MFIs are classified into two groups. One is the group of algorithms that directly find MFIs: LCMmax [1], Mafia [15] and GenMax [16]. The other is the group of algorithms that solves the dual problem, i.e., the problem of enumerating minimal infrequent itemsets: All_MSS [9] and IBE [17]. In MFI-mining, the algorithms in the former group are faster than the algorithms in the latter group according to the contest results [17, 18, 1]. So, the algorithms for the MEPSs corresponding to those in the former group are expected to be also faster. We propose an algorithm called LCMmax.R for MEPSs based on LCMmax, which is one of the fastest algorithms for MFIs. For the empirical comparison, we also implemented an algorithm called IBE.R based on IBE, which is one of the practically fastest algorithms in the latter group. (See Appendix A for the algorithm's details.)

LCMmax.R$_{\text{naive}}$(X: (exclusive) positive subset,
C: set of positive instances to be added)
G: stack for already processed positive instances

1: $g \leftarrow$ size of G
2: $C' \leftarrow$ the set of positive instances x in C s.t. $X \cup \{x\}$ is exclusive
3: $X \leftarrow X \cup (C' \cap r(X))$, $C' \leftarrow C' \setminus X$
4: **if** $r(X) \cap G \neq \emptyset$ **then** //*closure of X is NOT ppc-extension*
return one of maximal exclusive subsets $X'(\supseteq X)$ of $X \cup C'$
5: **if** $C' = \emptyset$ **then**
6: **if** $X \cup \{x\}$ is NOT exclusive for any $x \in G$ **then** //*X is maximal*
output X
7: **return** X
8: **end if**
9: choose a positive instance $x^* \in C'$, $C' \leftarrow C' \setminus \{x^*\}$
10: $X_0 \leftarrow$ LCMmax.R$_{\text{naive}}(X \cup \{x^*\}, C')$, $X' \leftarrow X_0$
11: push x^* to G
12: **for each** positive instance $x \in C' \setminus X_0$ **do**
13: $C' \leftarrow C' \setminus \{x\}$
14: $X'' \leftarrow$ LCMmax.R$_{\text{naive}}(X \cup \{x\}, C')$
15: push x to G
16: **if** $|X''| > |X'|$ **then** $X' \leftarrow X''$
17: **end for**
18: resize G as g //recover as previous state
19: **return** X'

Figure 1: Pseudo code of LCMmax.R$_{\text{naive}}$

4.2. Naive Version of LCMmax.R

LCMmax [1] is a recursive procedure that finds MFIs through depth first search by traversing only *closed* itemsets using a technique called *ppc(prefix preserving closure)-extension*. Different from Mafia and GenMax, LCMmax checks maximality of a found frequent itemset without using the MFIs found so far, which is not only more space-efficient but also faster when the number of MFIs is large.

By converting LCMmax simply using the correspondence described in the previous subsection, we can obtain a naive version of LCMmax.R, or LCMmax.R$_{\text{naive}}$, whose pseudo code is shown in Figure 1.

The algorithm takes as input an exclusive positive subset X and a set C of positive instances to be added to X. First, LCMmax.R$_{\text{naive}}$ calculates the set C' of positive instances $x \in C$ that can be used to extend X keeping exclusiveness, i.e., the set of positive instances $x \in C$ such that $X \cup \{x\}$ is also exclusive (Line 2). Next, some $x^* \in C'$ is chosen and LCMmax.R$_{\text{naive}}$ searches for the MEPSs that contain $X \cup \{x^*\}$ (Line 9-10). Let X_0 be one of the maximum-sized MEPS including $X \cup \{x^*\}$ (Line 10). Then, positive instances $x \in C'$ to be added to X next are narrowed down to the ones in $C' \setminus X_0$ (Line 12-17), which reduces the search space significantly if $|C' \cap X_0|$ is large. When any further extension is impossible for X, i.e., C' is empty at Line 5, LCMmax.R$_{\text{naive}}$ checks maximality of X (Line 6) and output it if it passed the test. Note that stack G is used for ppc-extension, and all the already processed positive instances are stored in G. For a positive subset X, the closure of X, or $r(X)$, is not a ppc-extension of X if $r(X)$ contains some already processed instances, i.e., $r(X) \cap G \neq \emptyset$. The non-ppc-extension $r(X)$ of X has been already processed as a ppc-extension of an other set before, so no further process is necessary (Line 4).

For simplicity, we omitted the description of the following two things used in our implementation in Figure 1. For efficient execution of the exclusiveness checks, $2d$ boundary values of $r(X)$ are also passed to the recursive calls. In order to enhance effect of the search tree pruning, namely, to get X_0 with large $|C' \cap X_0|$, x^* is selected so as to maximize the number of instances included in $r(X \cup \{x^*\})$ at Line 9, which can be calculated in $O(d|C'|^2)$ time.

The time complexity of LCMmax.R$_{\text{naive}}$ linearly depends on the number of the closed exclusive positive subsets (CEPSs). This is because Lines 5-19 are executed only for distinct CEPSs. Let n denote the number of the CEPSs

LCMmax.R(X: (exclusive) positive subset,
C: list of positive instances to be added)
G: stack for already processed positive instances
1-8: the same as Line 1-8 of LCMmax.$\mathrm{R}_{\mathrm{naive}}$ in Figure 1
9: sort C' in the decreasing order of $|r(X \cup \{C'[k]\}) \cap C'|$
10: $x^* \leftarrow C'[1]$, $C' \leftarrow C' \setminus \{x^*\}$ //$C'[1]$ is the head of C'
11: $X' \leftarrow$ LCMmax.R($X \cup \{x^*\}, C'$)
12: push x^* to G
13: move $C' \cap X'$ to the tail of C' while keeping the order within $C' \setminus X'$ and $C' \cap X'$,
$j \leftarrow |C' \setminus X'|$
14: initialize $c[C'[1]], \ldots, c[C'[j]]$ to 0 //counters for excluded frequency
15: **for** $i = 1, \ldots, j$ **do**
16: $x \leftarrow C'[i]$
17: $X'' \leftarrow$ LCMmax.R($X \cup \{x\}, C'[i+1..|C'|]$)
18: push x to G
19: **if** $|X''| > |X'|$ **then** $X' \leftarrow X''$
20: $c[C'[k]] \leftarrow c[C'[k]] + 1$ for all $C'[k] \notin X''$ and $k \in [i+1, j]$
21: sort $C'[i+1..j]$ in the decreasing order of $c[C'[k]]$ while keeping the order within the same valued instances
22: **end for**
23: resize G as g //recover as previous state
24: **return** X'

Figure 2: Pseudo code of LCMmax.R. Note that $C'[i..j]$ denotes the sublist of C' from the i-th element to the j-th element.

and let m^+ and m^- denote $|S^+|$ and $|S^-|$, respectively. For each CEPS, the maximality check (Line 6) and the selection of x^* (Line 9) are executed in $O(dm^+m^-)$ and $O(d(m^+)^2)$ time, respectively. Except the first execution, Lines 1-4 are executed by recursive calls at Line 10 and Line 14. For each CEPS, these recursive calls are executed at most m^+ times, so Lines 1-4 are also executed at most m^+ times. The selection of C' (Line 2) and the ppc-extension check (Line 4) are executed in $O(dm^+m^-)$ and $O(dm^+)$ time, respectively. Thus, in total, LCMmax.$\mathrm{R}_{\mathrm{naive}}$ takes $O(d(m^+)^2m^-n)$ time.

4.3. LCMmax.R

One of our proposed algorithms is LCMmax.R (Figure 2), which is an improved version of LCMmax.$\mathrm{R}_{\mathrm{naive}}$ described in the previous section. Two improvements have been implemented in LCMmax.R: dynamic reordering using excluded frequency and bit-parallel exclusiveness check. We describe the details of the two improvements below.

4.3.1. Effective dynamic reordering using excluded frequency

To reduce search space, the process order of instances in the for-loop (Line 15-22) is very important. In a good order, the set of instances that can be used to extend X keeping exclusiveness, i.e., the set C', which is calculated at Line 2 and 3, becomes smaller in average. If x is the i-th instance in C', which is denoted by $C'[i]$, then the preceding $i-1$ instances have been removed from the search space for $X \cup \{x\}$ (Line 17). In order to use this benefit effectively, the instances x that cannot reduce the search space very much, namely, x with large $\{y \in C' : y \notin r(X \cup \{x\})$ and $r(X \cup \{x, y\}) \cap S^- = \emptyset\}$, should be listed after the other instances with larger reduction of the search space.

There are two types of instances that are removed from C' by adding x to X: the instances y for which $X \cup \{x, y\}$ is NOT exclusive (Line 2), and the instances y that are contained in the closure of $X \cup \{x\}$ (Line 3). As for the second type of reduction, it is easy to calculate the effect of this reduction for each $x \in C'$, i.e., the number of instances $y \in C'$ in the closure of $X \cup \{x\}$. Thus, LCMmax.R sorts the elements x in C' in the decreasing order of the effect (Line 9). Unfortunately, it is too costly to calculate the effect of the first type of reduction for each instance $x \in C'$. Instead of calculating the amount of reduction exactly, we took the following strategy.

MRF(X, C)(X: (exclusive) positive subset,
C: list of positive instances to be added)
G: stack for already processed positive instances
M: current maximum-sized set
1-3: the same as Line 1-3 of LCMmax.R$_{\text{naive}}$ in Figure 1
4: **if** $|X| + |C'| \leq |M|$ **or** //*pruning by size*
5: $r(X) \cap G \neq \emptyset$ **then** //*closure of X is NOT ppc-extension*
return one of maximal exclusive subsets $X'(\supseteq X)$ of $X \cup C'$
6: **if** $C' = \emptyset$ **then**
7: $M \leftarrow X$, **return** X
8: **end if**
9-24: the same as Line 9-24 of LCMmax.R in Figure 2

Figure 3: Pseudo code of MRF

The number of instances y for which $X \cup \{x, y\}$ is NOT exclusive is upper-bounded by the number of times where X'' that excludes x is returned from the recursive call at Line 17. Thus, for each non-processed $x \in C'$, we count the number of times $c[x]$ where X'' excludes x (Line 20), and sort the non-processed instances x in C' in the decreasing order of $c[x]$ (Line 21). Note that we sort C' keeping the order within the same valued instances to maintain the effect of the sorting at Line 9 for the second type of reduction.

4.3.2. Bit-parallel exclusiveness check

The exclusiveness check takes $O(d|S^-|)$ time and is done so many times: $|C'|$ times at Line 2 and $|G|$ times at Line 6. Thus, the reduction of computational time in the exclusiveness check is very effective to reduce the total running time. LCMmax.R efficiently checks the exclusiveness of a set using bit-parallel processing.

Let $Y_i(a, b)$ denote the set of the negative instances in the interval $[a, b]$ on the i-th axis, that is, $Y_i(a, b) = \{(y_1, y_2, ..., y_d) \in S^- : y_i \in [a, b]\}$. Then, a positive subset X is exclusive iff $\bigcap_{i=1}^{d} Y_i(a_i, b_i) = \emptyset$ holds, where $\prod_{i=1}^{d}[a_i, b_i] = r(X)$. To calculate $\bigcap_{i=1}^{d} Y_i(a_i, b_i)$ efficiently, we represent $Y_i(a_i, b_i)$ by one $|S^-|$-bit sequence $B_i(a_i, b_i)$ in which the jth bit is 1 iff the jth negative instance belongs to $Y_i(a_i, b_i)$. Then, whether $\bigcap_{i=1}^{d} Y_i(a_i, b_i) = \emptyset$ or not can be checked by whether $\bigwedge_{i=1}^{d} B_i(a_i, b_i) = 00\ldots0$ or not. By this method, using k-bit CPU, it is checked parallely for k negative instances y whether $y_i \in [a_i, b_i]$ or not. For each ith axis, we have to prepare bit sequences $B_i(a_i, b_i)$ for all possible combinations of (a_i, b_i). Since we need the boundary values of $r(X)$ for positive subsets X only, $|S^+|(|S^+| - 1)/2$ sequences are enough, but they need much memory and preparation time. Thus, we propose a way in which at most $|S^+|$ sequences $B_i(a_i, \infty)$ and at most $|S^+|$ sequences $B_i(-\infty, b_i)$ are prepared for each ith axis, and the exclusiveness of a positive subset X is checked by whether $\bigwedge_{i=1}^{d} B_i(a_i, \infty) \wedge B_i(-\infty, b_i) = 00\ldots0$ or not, where $\prod_{i=1}^{d}[a_i, b_i] = r(X)$.

4.4. MRF

Compared with the enumeration of MEPSs, finding a maximum-sized MEPS is much easier task. Without the enumeration, a greedy cover $\{X_1, X_2, ..., X_k\} \subseteq \Omega(S^+, S^-, \mathcal{R})$ of S^+ can be obtained by the repetition of this easier task, that is, by executing the following procedure for $i = 1, 2, ..., k$: (1) finding a maximum-sized set $X'_i \in \Omega(S^+ \setminus \bigcup_{j=1}^{i-1} X_j, S^-, \mathcal{R})$ and (2) extending X'_i to a set $X_i \in \Omega(S^+, S^-, \mathcal{R})$ adding as many instances in $\bigcup_{j=1}^{i-1} X_j$ as possible while keeping the exclusiveness of the extended set. In this approach, we have to find a maximum-sized set $X'_i \in \Omega(S^+ \setminus \bigcup_{j=1}^{i-1} X_j, S^-, \mathcal{R})$ for all $i = 1, 2, ..., k$, but the task becomes easier for larger i because the number of the remaining uncovered positive instances becomes smaller. Thus, this approach without enumeration can be practically faster than the enumeration approach.

We developed an algorithm for finding a maximum-sized MEPS, which is called *MRF (Maximum Rectangle Finder)*. (See Figure 3.) MRF can be obtained by modifying LCMmax.R so as to find only one maximum-sized MEPS. MRF prunes the search space using the size of a maximum-sized MEPS M found so far (Line 4): skip the further search when the possible largest size of current extension, i.e., $|X| + |C'|$, is at most $|M|$. The reordering of instances $x \in C'$ by size $|r(X \cup \{x\}) \cap C'|$ at Line 9 (Figure 2) can accelerate the effect of this pruning because large

Table 1: Datasets used in our experiments. #MEPS is the number of MEPSs. "—" means unobtainable by 1 day calculation.

data	#label	#instance	#attribute	#MEPS
iris	3	150	4	30
glass	7	214	9	2600
haberman	2	306	3	400
balance	3	625	4	153
ecoli	8	336	7	606
heart-st	2	270	13	48732
tae	3	151	5	145
lung	3	32	56	77
liver	2	345	6	27435
wine	3	178	13	1058
ionosphere	2	351	34	—
vehicle	4	946	18	1796900
diabetes	2	768	8	1583574

MEPSs are expected to be found in the early stages of the search. Note that maximality check, which is done at Line 6 in LCMmax.R when $C' = \emptyset$, is nolonger necessary because X is always maximal at Line 7 in MRF, that is, $|X| + |C'| < |M|$ holds in the case with non-maximal X when $C' = \emptyset$.

MRF is surely faster than LCMmax.R owing to additional pruning by the maximum size among the sizes of already found MEPSs. How faster MRF is, however, depends on datasets, and so does whether RGC construction by repeated execution of MRF (MRF repetition) is faster than LCMmax.R. MRF repetition must execute MRF many times for the problems with a smaller set of positive instances, so MRF must solve such smaller problems significantly faster. Here, we analyze the dependence of time complexity on the number of positive instances in the worst case. We also analyze its empirical dependence in the next section.

According to the analysis of LCMmax.R_{naive} in Sec. 4.2, the time complexity of LCMmax.R_{naive} is $O((m^+)^2 n)$ with respect to the numbers m^+ and n of positive instances and closed exclusive positive subsets (CEPSs), respectively. The number n of CEPSs heavily depends on m^+, and is upper-bounded by $O((m^+)^{2d})$, where d is the number of attributes. Since the dependency on m^+ in the worst case does not seem to change even for the time complexities of LCMmax.R and MRF, the time complexity of MRF is $O((m^+)^{2d+2})$, which indicates that MRF runs significantly faster for the problems with a smaller set of positive instances.

5. Experiments

We conducted three experiments using benchmark datasets in UCI repository [11]. One is an experiment for checking computational efficiency of our greedy covering algorithms. The other two are experiments for demonstrating usefulness of classifiers using RGCs: comparisons with other classifiers in terms of classification performance and readability of classification rules.

5.1. Experimental Settings

We used 13 datasets of UCI repository [11] which are composed of real and integer attributes alone. The datasets used in our experiments are shown in Table 1. All the algorithms used in our experiments were implemented using C++ language. The experiments in Sec. 5.2.1, 5.4 and the first half of Sec. 5.2.2 were conducted on a machine with Opteron CPU (2.1GHz) and 4GB memory, and the rest experiments were conducted on a Dell Vostro with Intel(R) Core(TM)2Duo CPU E7500 (2.93GHz) and 4GB memory.

Table 2: Running time [sec] of the four algorithms for the six datasets.

	naive	DR	BE	DR+BE
glass	8.14	3.63	2.81	1.36
balance	2.98	1.14	0.66	0.26
ecoli	19.37	7.4	4.95	1.87
heart-st	1395.52	452.56	389.66	137.33
liver	100.36	44.8	25.45	12.72
wine	219.24	116.48	84.35	45.63

naive: LCMmax.R_{naive} (LCMmax.R without the two improvements),
DR: LCMmax.R with the dynamic reordering using excluded frequency only,
BE: LCMmax.R with the bit-paralleled exclusiveness check only,
DR+BE: LCMmax.R (with both of the two improvements).

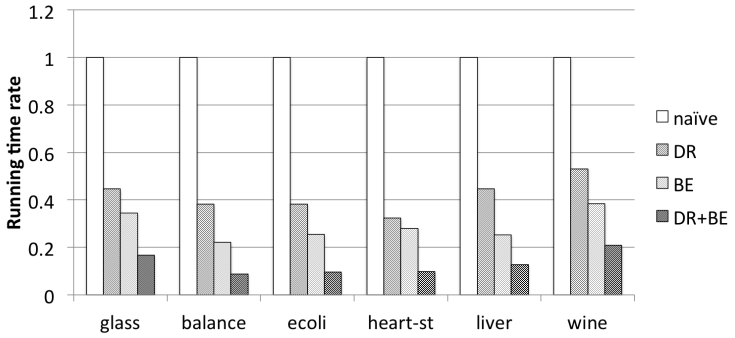

Figure 4: Running time which is normalized to that of LCMmax.R_{naive}. See Table 2 for notations 'naive', 'DR', 'BE' and 'DR+BE'.

5.2. Computational Efficiency

5.2.1. Effect of the Improvements Incorporated into LCMmax.R

We conducted an experiment in order to demonstrate the effect of the two improvements incorporated into LCMmax.R: dynamic reordering using excluded frequency and bit-parallel exclusiveness check. We measured running time of two variations of LCMmax.R, i.e., LCMmax.R with one of the two improvements only, in addition to that of LCMmax.R_{naive} (= LCMmax.R without the two improvements) and LCMmax.R (with both of the two improvements). We used 6 datasets for which LCMmax.R enumerates MEPSs spending 0.1-200 seconds.

The result is shown in Table 2. You can see that both improvements contribute to reducing the computation time. Running time rate shown in Figure 4 is the time normalized to the running time of LCMmax.R_{naive}. The bit-paralleled exclusiveness check makes the algorithm 2.6-4.5 times faster, and the dynamic reordering using excluded frequency makes it 1.9-3.1 times faster. Totally, LCMmax.R is 4.8-11.1 times faster than LCMmax.R_{naive}, which indicates that the effects of the two improvements are independent.

5.2.2. Comparison of Three Construction Algorithms for an RGC

We empirically compared efficiency of the construction algorithms for an RGC. Three algorithms we compared in our experiment are IBE.R (Figure A.6), LCMmax.R (Figure 2), and MRF repetition, namely, repeated execution of MRF (Figure 3). The first two algorithms enumerate MEPSs and need an independent greedy covering process using found MEPSs to obtain an RGC. In our experiment, we measured the running time of enumeration process only for those algorithms. As for MRF repetition, the running time that is spent by all the processes until an RGC is obtained, is measured. For datasets with k classes, the algorithms were executed k times choosing one different class as the class of positive instances and regarding the instances of all the other classes as negative ones. Then, the running time was summed up over all the k executions.

Table 3: Running time [sec] spent by IBE.R, LCMmax.R, and MRF repetition. "< 0.01" means less than 0.01 second and ">1day" means more than one day.

	IBE.R	LCMmax.R	MRF rep.
iris	0.01	0.01	**< 0.01**
glass	53.56	1.36	**0.18**
haberman	**< 0.01**	0.03	0.05
balance	1.55	0.26	**0.24**
ecoli	26.43	1.87	**0.40**
heart-st	> 1day	137.33	**9.02**
tae	0.01	**< 0.01**	**< 0.01**
lung	**< 0.01**	0.01	**< 0.01**
liver	28875.00	12.72	**3.03**
wine	1110.00	45.63	**1.15**
ionosphere	> 1day	> 1day	**792.05**
vehicle	> 1day	71770.06	**565.79**
diabetes	> 1day	73940.17	**904.04**

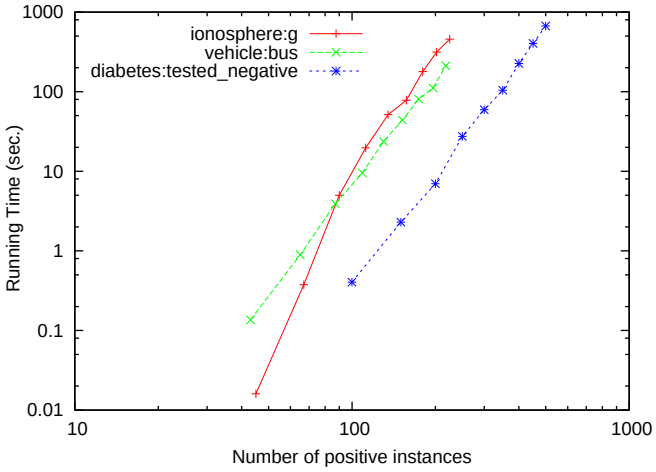

Figure 5: Running time dependency of MRF on the number of positive instances

The result is shown in Table 3. As for algorithms using the enumeration approach, LCMmax.R was faster than IBE.R for 11 datasets among the 13 datasets. The running time difference between the two algorithms increases when the number of rectangles to be enumerated becomes larger; LCMmax.R was 2270 time faster than IBE.R for liver dataset (27435 MEPSs), and even for the datasets with more MEPSs, LCMmax.R finished within one day except the ionosphere dataset, while IBE.R could not finish within one day for any of the datasets. Compared with the two enumeration algorithms, MRF repetition, an algorithm using the direct (non-enumeration) approach, was faster for all the datasets but the haberman dataset. Especially for the datasets with a large number of MEPSs, MRF repetition runs significantly faster than LCMmax.R, for example, 127 times faster for the vehicle dataset.

In order to investigate the fastness of MRF repetition, we further conducted an experiment on running time dependency of MRF on the number of positive instances. Setting each maximum-sized class of the ionosphere, vehicle and diabetes datasets to the positive class, we ran MRF on these three datasets using only the first 100α % of the positive instances for $\alpha = 0.2, 0.3, ..., 1.0$. The result is shown in the graph of Figure 5, in which both axes are logarithmic scales. The slopes of the broken lines for each dataset look stable for vehicle and diabetes datasets and even for ionosphere dataset with respect to the right-most five points. So, we checked the empirical time complexity of MRF for each dataset by fitting function f ($f(m^+) = a(m^+)^b$). We applied the least square method to the linear function $y' = bx' + \log_{10} a$ using the rightmost five points $(x', y') = (\log_{10} x, \log_{10} y)$ for the number x of positive instances and

its corresponding running time y. The result is shown in Table 4.

Table 4: Running time of MRF and its empirical time complexity with respect to the number of positive instances

data	ionosphere	vehicle	diabetes
positive (#)	class=g (225)	Class=bus (218)	class=tested_negative (500)
RGC size	4	5	26
maximum size	179	119	140
running time	456.36	211.12	668.65
(covering time)	(459.25)	(217.93)	(854.95)
fitting function $a(m^+)^b$	$a = 1.14 \times 10^{-8}$ $b = 4.51$	$a = 4.83 \times 10^{-8}$ $b = 4.11$	$a = 5.18 \times 10^{-11}$ $b = 4.86$

The empirical exponent b of m^+ is more than 4 for all the three datasets. Though these values are small compared to $2d + 2$ in the theoretical worst-case upper-bound, they are still large, where d is the number of attributes. For ionosphere, vehicle and diabetes datasets, the first execution of MRF reduces their numbers of positive instances to 20%, 45% and 72%, respectively, of them, which expectedly results in reducing their running times of the next execution to 0.07%, 3.5% and 20%, respectively, of those of the first execution. Therefore, most running time of MRF repetition is consumed by the first execution of MRF in these datasets, which means that running-time reduction of MRF from LCMmax.R reduces the running time of MRF repetition by a similar factor.

In the following experiments, we used MRF repetition to obtain an RGC.

5.3. Classification performance

First, we compared the classification performance of RGC to that of Randomized Subclass Method (RSM) [2], which is a conventional method that constructs a set of rectangles $\{r(X_1), r(X_2), ..., r(X_k)\}$ by repeatedly finding an MEPS $X_i \in \Omega(S^+, S^-, \mathcal{R})$ not greedily but randomly[4] for given S^+ and S^-; one MEPS X is found randomly using random permutation $x_1, x_2, ..., x_n$ of the elements in S_+ by initializing X to $\emptyset$ and adding x_i to X in the order of the permutation if the exclusiveness of X is preserved. The number of rectangles k that are constructed by RSM is a parameter we have to specify. In our experiments, RSM was modified to continue to find a rectangle until S^+ is covered.

As a classification method using rectangles, a nearest rectangle classifier [5] is used. As a distance of an instance from a hyperrectangle, we simply used Euclidean distance from the nearest boundary of the hyperrectangle if the instance is outside the hyperrectangle and 0 otherwise[5]. For multi-class problem, MEPSs were generated for each class by regarding the class as positive class and all the other classes as one negative class. We applied majority voting over the classes of the nearest rectangles when more than one nearest rectangle exists[6]. If there is more than one class that has the same number of the nearest rectangles, the instance is classified into the class that is randomly chosen from those classes.

Classification accuracy was calculated by 10-fold cross validation[7].

According to the result shown in Table 5, classification accuracy of RGC is a little worse than that of RSM. RGC is considered to have larger bias than RSM because of its simplicity, which might cause the lower accuracy. As

[4]In our implementation, the function *rand* of the standard library of C language is used for randomization. No seed is given by the *srand* function, so the generated MEPS sequence is always the same for the same data.

[5]In the distance used in [5], hyperrectangles and features are weighted and each feature is also normalized by the difference between its maximum and minimum values. In our experiments, no weighting and no normalization were applied to the distance.

[6]The case includes the situation in which the instanace is included in rectangles of different classes. In that situation, in [5], the instance is classified into the class of the smallest hyperrectangle that contains the instance, though classification was done by majority voting even in such situation in our experiments.

[7]The cross-validation folds were generated by using Weka's API for fare comparison with the results of the later experiments using Weka Explorer. See http://weka.wikispaces.com/Generating+cross-validation+folds+(Java+approach).

Table 5: Classification accuracy compared to other algorithms[%].

	RGC	RSM	AB	L. R.	N. B.	SVM	C4.5	kNN
iris	95.33	*94.67*	95.33	**96.00**	**96.00**	**96.00**	**96.00**	95.33
glass	69.16	67.76	*44.86*	63.55	48.60	67.76	66.82	**69.63**
haberman	*65.36*	67.65	74.18	74.18	**76.14**	73.53	72.88	71.90
balance	84.48	87.04	*72.32*	89.60	90.40	**90.88**	76.64	89.60
ecoli	79.76	84.23	*64.58*	**86.61**	85.42	74.11	84.23	87.50
heart-st	78.52	82.22	80.37	**83.70**	**83.70**	*64.81*	76.67	78.89
tae	62.91	**66.89**	*37.75*	54.30	54.30	54.30	59.60	62.25
lung	**68.75**	62.50	46.25	50.00	62.50	50.00	50.00	*40.63*
liver	66.96	71.01	**73.33**	68.12	*55.36*	68.70	68.70	61.74
wine	93.82	93.82	91.01	98.31	**96.63**	*67.98*	93.82	96.07
ionosphere	91.74	91.17	92.59	88.89	*82.62*	**93.45**	91.45	88.60
vehicle	72.10	72.10	*39.95*	**79.79**	44.80	69.62	72.46	69.50
diabetes	*71.61*	76.43	75.00	**77.21**	76.30	71.74	73.83	72.79
ave.	76.96	**78.27**	*68.27*	77.71	73.29	72.53	75.62	75.73

AB: AdaBoost (*component classifier*: decision stumps)
L. R.: logistic regression with a ridge estimator
N. B.: Naive Bayes
SVM: (*software*: LibSVM, *kernel*: RBF-kernel $K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma|\boldsymbol{x}-\boldsymbol{y}|^2}$)
kNN: k-nearest neighbor

for simplicity of the classifiers, see the column #terms in Table 6, in which the number of rectangles generated by the classifiers is shown. The number of rectangles contained in RGC is, in average, about 3.5 times smaller than the number of rectangles generated by RSM, and 8.8 times smaller in the best case (the case with heart-st dataset). Thus, RGC enables not only more time- and space-efficient prediction but also better performance for sparse datasets, namely, the datasets with a small number of instances and a large number of attributes. In fact, RGC outperformed RSM for lung dataset. Furthermore, RGC is more stable than RSM considering the fact that a set of rectangles generated by RSM depends on the random numbers it used.

Next, we compared the classification accuracy of RGC to that of 6 popular classifiers: AdaBoost, logistic regression with a ridge estimator, Naive Bayes, SVM, C4.5 (J48), and kNN (k-nearest neighbor). In the experiment, we used the softwares of those 6 classifiers provided by Weka [20]. The number I of iterations of AdaBoost, the ridge parameter R of logistic regression, the gamma parameter γ of RBF-kernel of SVM and the parameter k of kNN were selected by grid search with cross validation evaluation using training data only: the tried values were $I = 30, ..., 300$ with 10 steps, $R = 10^{-9}, ..., 10^{-6}$ with 100 steps, $\gamma = 0.001, ..., 0.1$ with 100 steps and $k = 1, ..., 10$ with 10 steps. The other parameter values of those softwares used in our experiment were set to the default values. Note that RGC doesn't have any parameters to tune.

We calculated classification accuracy of the above classifiers by 10-fold cross validation using Weka Explorer.

The result is also shown in Table 5. RGC outperformed 5 popular classifiers (AdaBoost, Naive Bayes, SVM, C4.5 and kNN) in average. RGC also outperformed logistic regression for 4 datasets, and its average accuracy was comparable to that of logistic regression.

5.4. Readability of Classification Rules

We compared the rules generated by RGC with those by RSM and C4.5 [12] using the whole datasets. In order to compare with the rules generated by RGC and RSM, which are 100%-accurate for the input dataset, the parameters of C4.5 (J48) were set so as to maximize accuracy of the rules for the input dataset: the minimum number of instances per leaf was set to 1 and the unpruned option was selected. The decision trees obtained by C4.5 were transformed into corresponding DNF-rules for comparison.

Table 6 shows the number of terms and the number of atoms in the DNF rules generated by the three classifiers for our first 10 datasets, for which RGC rules could be calculated within 10 seconds. Here, an atom is a representation

Table 6: the number of terms and atoms.

	#terms			#atoms			accuracy
	RGC	RSM	C4.5	RGC	RSM	C4.5	C4.5
iris	**8**	12.6	10	**21**	35.2	29	1.00
glass	**22**	88.8	37	**99**	574.3	216	0.97
haberman	63	227.7	**13**	208	951.4	**45**	0.80
balance	**153**	**153.0**	161	**556**	**556.0**	978	1.00
ecoli	**33**	119.1	34	**135**	622.9	169	0.95
heart-stat	**19**	167.6	51	**120**	1092.5	318	1.00
tae	**38**	84.2	47	**129**	414.8	268	0.95
lung	**6**	8.0	13	**30**	46.0	62	1.00
liver	36	263.5	**34**	193	1690.3	**170**	0.85
wine	**5**	7.0	7	**14**	35.0	21	0.99

$x \leq a$, $x \geq a$, $x > a$, or $x = a$ for an attribute x and a real value a, and a term is a conjunction of atoms. The DNF-rules obtained by RGCs are simpler than those obtained by RSM and C4.5 except for haberman and liver datasets. Note that the accuracy of C4.5 for the two datasets are less than 90%, which enabled smaller-sized representation to be generated.

The DNF-representation of rules generated by an RGC for Iris-versicolor class of the iris dataset is

$$\begin{aligned}&(\text{petal-length} \leq 4.9 \wedge \text{petal-wid} \in [1, 1.6]) \vee\\ &(\text{petal-length} \in [5, 5.1] \wedge \text{petal-wid} \in [1.6, 1.7]) \vee\\ &(\text{setal-wid} \geq 3.2 \wedge \text{petal-length} \leq 4.8 \wedge \text{petal-wid} \geq 1.8),\end{aligned}$$

where len, and wid are abbreviations of length and width, respectively. Note that boundaries, which can be removed while keeping exclusiveness, are removed as described in Sec. 3.3.

The corresponding representation constructed by RSM is

$$\begin{aligned}&(\text{petal-len} \leq 4.9 \wedge \text{petal-wid} \in [1, 1.6]) \vee\\ &(\text{setal-len} \geq 5 \wedge \text{setal-wid} \geq 2.3 \wedge \text{petal-len} \leq 5 \wedge \text{petal-wid} \in [1, 1.7]) \vee\\ &(\text{setal-wid} \in [2.3, 2.7] \wedge \text{petal-len} \leq 5.1 \wedge \text{petal-wid} \in [1, 1.6]) \vee\\ &(\text{setal-len} \leq 6.2 \wedge \text{setal-wid} \geq 2.3 \wedge \text{petal-len} \leq 5.1 \wedge \text{petal-wid} \in [1, 1.6]) \vee\\ &(\text{setal-wid} \geq 3.1 \wedge \text{petal-len} \leq 4.9 \wedge \text{petal-wid} \geq 1.4),\end{aligned}$$

and that constructed by C4.5 is

$$\begin{aligned}&(\text{petal-len} \leq 4.9 \wedge \text{petal-wid} \in [0.6, 1.5]) \vee\\ &(\text{petal-len} \leq 4.9 \wedge \text{petal-wid} \in [1.5, 1.6]) \vee\\ &(\text{setal-len} \leq 6.9 \wedge \text{petal-len} > 4.9 \wedge \text{petal-wid} \in [1.5, 1.7]) \vee\\ &(\text{setal-len} \leq 5.9 \wedge \text{petal-len} \leq 4.8 \wedge \text{petal-wid} > 1.7).\end{aligned}$$

As you can see, the classification rule for Iris-versicolor constructed by an RGC is much easier to understand than those constructed by RSM and C4.5.

6. Concluding Remarks

We developed efficient construction methods of an RGC and demonstrated its usefulness as a classifier and a readable DNF-rule.

RGCs have not been used practically by now because previous algorithms are too slow for practical use. MRF repetition is the first practical algorithm for RGC construction though its computational cost is still too high for large datasets.

The classification performance of a nearest neighbor rectangle classifier using an RGC was demonstrated to be close to that of conventional methods in spite of its representation simplicity, which enables efficient classification.

A simple DNF-representation of an RGC can well help us analyze a dataset in data mining.

In the future, a more efficient algorithm is required for applying it to large datasets, and a cover using non-exclusive rectangles is also desirable for obtaining simpler and more robust classification rules.

acknowledgement

This work was partially supported by JSPS KAKENHI 21500128.

References

[1] T. Uno, M. Kiyomi, H. Arimura, LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets, in: IEEE ICDM'04 Workshop FIMI'04, 2004.
[2] M. Kudo, S. Yanagi, M. Shinbo, Construction of class regions by a randomized algorithm: A randomized subclass method, Pattern Recognition 29 (1996) 581–588.
[3] K. Ouchi, A. Nakamura, M. Kudo, Efficient construction and usefulness of hyper-rectangle greedy covers, in: 2011 IEEE International Conference on Granular Computing, 2011, pp. 533–538.
[4] M. Kudo, M. Shimbo, Optimal subclasses with dichotomous variables for feature selection and discrimination, IEEE Transactions on Systems, Man, and Cybernetics 19 (1989) 1194–1199.
[5] S. Salzberg, A nearest hyperrectangle learning method, Machine learning 6 (3) (1991) 251–276.
[6] D. Wettschereck, T. Dietterich, An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms, Machine Learning 19 (1) (1995) 5–27.
[7] Y. Shidara, M. Kudo, A. Nakamura, Extraction of generalized rules with automated attribute abstraction, Foundations of Data Mining and knowledge Discovery, Studies in Computational Intelligence 6 (2005) 161–170.
[8] A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth, Learnability and the vapnik-chervonenkis dimension, Journal of the ACM (JACM) 36 (4) (1989) 929–965.
[9] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, R. Sharma, Discovering all most specific sentences, ACM Transactions on Database Systems (TODS) 28 (2) (2003) 140–174.
[10] A. Nakamura, M. Kudo, What sperner family concept class is easy to be enumerated?, Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (2008) 482–491.
[11] C. L. Blake, C. J. Merz, UCI repository of machine learning databases, [`http://www.ics.uci.edu/~mlearn/mlrepository.html`] (1998).
[12] J. Quinlan, C4. 5: programs for machine learning, Morgan Kaufmann, 1993.
[13] C. Lund, M. Yannakakis, On the hardness of approximating minimization problems, Journal of the ACM (JACM) 41 (5) (1994) 960–981.
[14] M. Marchand, J. Shawe-Taylor, The set covering machine, The Journal of Machine Learning Research 3 (2003) 723–746.
[15] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, T. Yiu, Mafia: A maximal frequent itemset algorithm, Knowledge and Data Engineering, IEEE Transactions on 17 (11) (2005) 1490–1504.
[16] K. Gouda, M. Zaki, Genmax: An efficient algorithm for mining maximal frequent itemsets, Data Mining and Knowledge Discovery 11 (3) (2005) 223–242.
[17] T. Uno, K. Satoh, Detailed description of an algorithm for enumeration of maximal frequent sets with irredundant dualization, in: Online CEUR Workshop Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 2003.
[18] T. Uno, T. Asai, Y. Uchida, H. Arimura, LCM: An efficient algorithm for enumerating frequent closed item sets, in: IEEE ICDM'04 Workshop FIMI'03, 2003.
[19] K. Murakami, T. Uno, Efficient algorithms for dualizing large-scale hypergraphs, Arxiv preprint arXiv:1102.3813 0 (0) (2011) 0.
[20] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. Witten, The WEKA data mining software: An update, ACM SIGKDD Explorations Newsletter 11 (1) (2009) 10–18.
[21] D. Avis, K. Fukuda, Reverse search for enumeration, Discrete Applied Mathematics 65 (1) (1996) 21–46.

Appendix A. IBE.R

IBE (Irredundant Border Enumerator [17]) is an improved algorithm of All_MSS [9]. These algorithms obtain a new member of a target maximal-set family by finding and expanding a seed set, which is a subset of an unknown member. Such seed set H can be always found among the minimal sets that are not included in any member M_i found so far, that is, H can be found among the minimal hitting sets $mhs(\{\overline{M_0}, \overline{M_1}, \ldots, \overline{M_{k-1}}\})$ of $\{\overline{M_0}, \overline{M_1}, \ldots, \overline{M_{k-1}}\}$, where $\{M_0, M_1, \ldots, M_{k-1}\}$ is the set of members found so far and $\overline{M_i}$ is the complement set of M_i. Note that a

$M_0, \ldots, M_{k-1}$: MEPSs found so far
IBE.R(i, H)
i: depth of recursive calls
H: a minimal hitting set of $\{\overline{M_0}, \overline{M_1}, \ldots, \overline{M_{i-1}}\}$

1: **if** $i = k$ **then**
2: **if** $r(H) \cap S^- \neq \emptyset$ **then return**
3: $M_k \leftarrow H$
4: make M_k maximal
by appending $x \in S^+$ with $r(M_k \cup \{x\}) \cap S^- = \emptyset$ to M_k
5: $k \leftarrow k + 1$.
6: **end if**
7: **if** $\overline{M_i} \cap H \neq \emptyset$ **then**
8: IBE.R$(i + 1, H)$
9: **else**
10: **for each** $x \in \overline{M_i}$ **do**
11: **if** $H \cup \{x\}$ is a minimal hitting set of $\{\overline{M_0}, \overline{M_1}, \cdots, \overline{M_i}\}$ **then**
12: IBE.R$(i + 1, H \cup \{x\})$
13: **end if**
14: **end for**
15: **end if**

Figure A.6: Pseudo code of IBE.R

hitting set of a set family $\mathcal{C}$ is a set that has non-empty intersection with any member of $\mathcal{C}$. To fond the seeds of any unknown members for the target maximal-set family $\mathcal{M} = \{M_0, M_1, \ldots, M_{n-1}\}$, IBE generates all the elements H of $\bigcup_{k=0}^{n} mhs(\{\overline{M_0}, \overline{M_1}, \ldots, \overline{M_{k-1}}\})$ without duplication using reverse search [21].

A pseudo code[8] of IBE.R, which is an MEPS-version of IBE, is shown in Figure A.6. The MEPSs can be enumerated by executing IBE.R$(0, \emptyset)$. Note that the exclusiveness checks at lines 2 and 4 are the replacements of the frequentness checks in the original IBE. In our implementation of IBE.R, $2d$ boundary values of $r(H)$ are also passed to the recursive calls because $r(H \cup \{x\})$ can be calculated from them in $O(d)$ time while it takes $O(d|H \cup \{x\}|)$ time without them.

[8]The faster algorithm can be implemented by using the recently developed algorithm in [19].