An Algorithm for Mining Weighted Dense Maximal 1-Complete Regions

Haiyun Bian and Raj Bhatnagar

Department of ECECS, University of Cincinnati, Cincinnati, OH 45221, USA
bianh@ececs.uc.edu, raj.bhatnagar@uc.edu

Summary. We propose a new search algorithm for a special type of subspace clusters, called maximal 1-complete regions, from high dimensional binary valued datasets. Our algorithm is suitable for dense datasets, where the number of maximal 1-complete regions is much larger than the number of objects in the datasets. Unlike other algorithms that find clusters only in relatively dense subspaces, our algorithm finds clusters in all subspaces. We introduce the concept of weighted density in order to find interesting clusters in relatively sparse subspaces. Experimental results show that our algorithm is very efficient, and uses much less memory than other algorithms.

1 Introduction

Frequency has been used for finding interesting patterns in various data mining problems, such as the minimum support threshold used in mining frequent itemsets [2,3] and the minimum density defined in mining subspace clusters [1]. A priori-like algorithms [1] perform levelwise searches for all patterns having enough frequencies (either support or density) starting from single dimensions, and prune the search space based on the rationale that in order for a $k-$dimensional pattern to be frequent, all its $(k-1)-$dimensional sub-patterns must also be frequent. A large frequency threshold is usually set in most of the algorithms to control the exponential growth of the search space as a function of the highest dimensionality of the frequent patterns.

Closed patterns was proposed [7] to reduce the number of frequent patterns being returned by the algorithm without losing any information. Mining closed patterns is lossless in the sense that all frequent patterns can be inferred from the set of closed patterns. Most algorithms proposed for mining closed patterns require all candidates found so far to be kept in memory to avoid duplicates [9, 11, 12]. These algorithms also require the minimum frequency threshold value to be specified before the algorithms are run, and the same value is used to prune off candidates for patterns in all subspaces.

H. Bian and R. Bhatnagar: *An Algorithm for Mining Weighted Dense Maximal* 1*-Complete Regions*, Studies in Computational Intelligence (SCI) **118**, 31–48 (2008)
www.springerlink.com

Table 1. Subspaces with varied density

	a	b	c	d	e	f
1	0	0	0	0	0	1
2	0	0	0	0	0	1
3	0	0	0	0	1	1
4	0	0	0	1	1	1
5	0	0	0	1	1	1
6	0	0	0	1	1	1
7	0	0	0	1	1	0
8	0	0	0	1	0	0
9	1	1	1	0	0	0
10	1	1	1	0	0	0

However, patterns with higher dimensionality tend to have less frequencies, so using the same threshold value for all patterns risks losing patterns in higher dimensional spaces. Furthermore, patterns with the same dimensionality may need different frequency threshold values for various reasons. For example, a pattern with higher frequency in very dense dimensions may not be as informative and interesting as a pattern with lower frequency in very sparse dimensions. Setting a relatively high frequency threshold tends to bias the search algorithm to favor patterns in dense subspaces only, while patterns in less dense subspaces are neglected. Consider the example shown in Table 1. Each column denotes one of the six attributes (a, b, c, d, e, f), and each row denotes one object (data point). An entry '1' in row i and column j denotes that object i has attribute j. There is a pattern in subspace $\{abc\}$ that contains two instances $\{9, 10\}$, and subspace $\{def\}$ has another pattern containing three instances $\{4, 5, 6\}$. If we set the minimum frequency threshold to be 3, we lose the pattern in $\{abc\}$. However, this pattern in $\{abc\}$ maybe more interesting than the one in $\{def\}$, considering the fact that the number of '1's in attributes a, b, c is much smaller than in attributes d, e, f. Actually, all instances that have entry '1' in a also have entry '1' in b and c, and this may suggest a strong correlation between a, b, c, and also a strong correlation between instances 9 and 10. On the other hand, although the pattern in $\{def\}$ has a larger frequency, it does not suggest such strong correlations either between attributes d, e, f or between instances 4–6. So we suggest that smaller frequency threshold should be chosen for subspaces with lower densities, that is, subspaces with less number of '1' entries.

We propose a weighted density measure in this chapter, which captures the requirement to use a smaller density threshold for less dense subspaces. And we present an efficient search algorithm to find all patterns satisfying a minimum weighted density threshold.

Most algorithms for finding closed patters report only the dimensions in which the patterns occur, without explicitly listing all the objects that are contained in the patterns. However, the object space of the patterns is crucial

in interpreting the relationships between two possibly overlapping patterns. Our algorithm treats the objects and dimensions (attributes) equally, and all patterns are reported with their associated dimensions and subsets of objects.

Another advantage of our algorithm lies in its step-wise characteristic, that is, the computation of the next pattern depends only on the current pattern. Our algorithm is memory efficient due to this property, since there is no need to keep all previously generated patterns in the memory.

In the rest of the chapter, we present our algorithm in the context of the subspace clustering problem, but the algorithm and the theorem can also be applied to other closed set mining problems such as frequent closed itemsets [7] and maximal biclique [8]. We first present in Sect. 2 the definition of maximal 1-complete region, where we also introduce the terms and notations used in this chapter. Section 3 presents our algorithm. Section 4 presents the experimental results. Finally, we make the conclusion.

2 Problem Statement

A data space $\mathcal{DS}$ is characterized by a set of attributes $\mathcal{A}$ (attribute space) and a population of objects $\mathcal{O}$ (object space). Each object $o_i \in \mathcal{O}$ has a value assigned for each attributes $a_j \in \mathcal{A}$, denoted as d_{ij}. We consider only binary valued datasets in this chapter, that is, $d_{i,j} \in [0, 1]$. However, real valued datasets can be quantized into binary values, and different quantization methods lead to clusters of different semantics [6]. A subspace S is a subset of $\mathcal{A}$. A subspace cluster C is defined as $<O, A>$, where $O \subseteq \mathcal{O}$ and $A \subseteq \mathcal{A}$. We call O and A the *object set* and the *attribute set* of the subspace cluster respectively. Subspace clustering is a search for subsets of $\mathcal{P}(\mathcal{A})$ (the power set of $\mathcal{A}$) where interesting clusters exist.

2.1 The Prefix Tree of Subspaces

Let "$<_L$" be a lexicographic order on the attributes in $\mathcal{A}$, and we use $a_i <_L a_j$ to indicate that attribute a_i is lexicographically smaller than attribute a_j. Each subspace is represented as the set of attributes contained in it in the lexicographically increasing order. For example, a subspace containing attribute a_1, a_2, a_3 ($a_1 <_L a_2 <_L a_3$) is labeled as $\{a_1a_2a_3\}$. And we arrange all subspaces into a prefix-based tree structure $\mathcal{T}_{\mathcal{DS}}$ as follows:

1. Each node in the tree corresponds to one subspace, and the tree is rooted at the node corresponding to the empty subspace that contains no attributes.
2. For a node with label $S = \{a_1, \ldots, a_{k-1}, a_k\}$, its parent is the node whose label is $S' = \{a_1, \ldots, a_{k-1}\}$.

Table 2 shows an example dataset, and Fig. 1 shows its prefix tree of subspaces.

Table 2. An example data table

	a	b	c	d
1	0	0	1	1
2	1	0	1	1
3	1	1	1	0
4	0	0	1	1
5	1	1	0	0
6	0	0	1	1
7	0	0	1	1
8	0	1	0	0

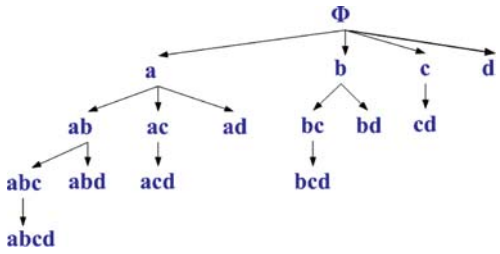

Fig. 1. Prefix-based subspaces search tree

2.2 Maximal 1-complete Regions and Closed Subspaces

We are interested in finding subspace clusters that contain largest regions of '1' entries, formally defined as follows:

Definition 1. *A subspace cluster* $C = <O, A>$ *of binary valued data space* $\mathcal{DS}$ *is a* 1-complete region *if it contains only '1' entries.*

Definition 2. *A complete dense region* $C = <O, A>$ *is a* maximal 1-complete region *if all regions that are proper super-regions of* C *are not* 1*-complete.*

For the example shown in Table 2, $<\{1, 2, 4, 6, 7\}, \{d\}>$ is a 1-complete region but it is not maximal, because its super-region $<\{1, 2, 4, 6, 7\}, \{cd\}>$ is 1-complete. $<\{1, 2, 4, 6, 7\}, \{cd\}>$ is a maximal 1-complete region, while $<\{1, 2, 3, 4, 6, 7\}, \{cd\}>$ is not 1-complete since it contains zero entries. If we consider each attribute (column) in Table 2 as a bit vector, all 1-complete regions can be found by intersecting all possible subsets of attributes. However, not all of them are maximal, so the problem is to find those subsets of attributes whose intersection produce maximal 1-complete regions.

Definition 3. *If a subspace is the attribute set of a maximal* 1*-complete region, we call this subspace a* closed subspace*.*

According on Definition 3, each maximal 1-complete region corresponds to one unique closed subspace. In order to find all maximal 1-complete regions, we can traverse the prefix tree of subspaces and check each node to see whether it is a closed subspace. In the following, we present several methods to test whether a subspace is closed. We first introduce two functions that perform mapping between the object space and the attribute space.

We define $\psi(S)$ to be $\{o_i|\forall a_j \in S, d_{ij} = 1\}$, that is, $\psi(S)$ returns the set of objects that have entry '1' for all the attributes in S. Similarly, $\varphi(O)$ is defined to be $\{a_j|\forall o_i \in O, d_{ij} = 1\}$, that is, $\varphi(O)$ returns the set of attributes that are shared by all objects in O. Then $\varphi \circ \psi$ is a closure operator, and we have the following lemma.

Lemma 1. *The following statements are equivalent:*

1. $C = <\psi(S), S>$ *is maximal* 1*-complete*
2. S *is a closed subspace*
3. $\nexists a \in \mathcal{A}/S$*, for which* $\psi(a) \supseteq \psi(S)$
4. $\varphi \circ \psi(S) = S$

Proof. $1 \leftrightarrow 2$: True by Definition 3.

$1 \rightarrow 3$: $C = <\psi(S), S>$ is maximal 1-complete means that we cannot add any attribute a to S to get an enlarged region, and at the same time maintain the 1-complete property. If there exists a for which $\psi(a) \supseteq \psi(S)$, then adding a to S will produce a region that has 1-complete property, which contradicts the fact that C is maximal 1-complete.

$3 \rightarrow 4$ and $4 \rightarrow 1$ can be proved similarly. □

From Lemma 1, we can see that $\varphi \circ \psi(S)$ is a superset of S if S is not closed, or equal to S if S is closed. Figure 2 shows a modified prefix tree from Fig. 1, where each node in Fig. 2 has two labels, including the corresponding subspace S and the object set $\psi(S)$. For example, node "$b, 358$" (Fig. 2) represents that this node corresponds to subspace $\{b\}$, for which $\psi(\{b\}) = \{3, 5, 8\}$. Underlined nodes are those 1-complete regions that are not maximal. Furthermore, nodes corresponding to subspaces with equal closure are grouped together into one equivalence class in Fig. 2. For example, $\varphi \circ \psi\{bc\} = \{abc\}$, so nodes "$bc$" and "$abc$" are grouped together. Notice that all equivalent subspaces have the same object set, so each equivalence class generates only one maximal 1-complete region. Therefore, we need only find one subspace for every such equivalence class in order to find all 1-complete regions.

2.3 The Lectical Order Between Subspaces

From Fig. 2, we can see that within each equivalence class, the closed subspace is always to the left of those non-closed ones. Based on this observation, we define a total order, called the *lectical* order, on the set of all subspaces. A similar definition can be found in [5]. A subspace S_1 is called *lectically smaller*

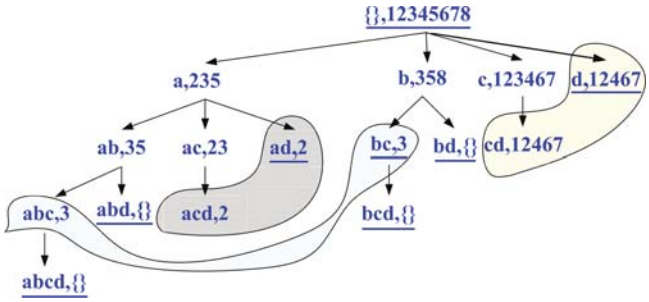

Fig. 2. Prefix tree of equivalence classes

than subspace S_2, denoted as $S_1 \ll S_2$, if the lexicographically smallest attribute a_i that distinguishes S_1 from S_2 belongs to S_2. That is, there exists $a_i \in S_2 \wedge a_i \notin S_1$, and all attributes lexicographically smaller than a_i are shared by S_1 and S_2. Formally,

$S_1 \ll S_2 :\Leftrightarrow \exists_{a_i \in S_2 \setminus S_1}\ S_1 \cap \{a_1, a_2, \ldots, a_{i-1}\} = S_2 \cap \{a_1, a_2, \ldots, a_{i-1}\}$.

If we know the attribute a_i that distinguishes S_1 and S_2, we say S_1 is i-smaller than S_2, denoted as $S_1 \ll_i S_2$.

For example, $\{ad\} \ll_c \{acd\}$ because the lexicographically smallest attribute that distinguishes them is c, and it belong to $\{acd\}$.

We define S^i to be a subset of S which includes all the attributes in S that are lexicographically smaller than a_i, that is, $S^i := S \cap \{a_1, \ldots, a_{i-1}\}$. Starting from an arbitrary subspace S, the next lectically smallest subspace that is larger than S can be computed based on Lemma 2.

Lemma 2. *The lectically smallest subspace that is lectically larger than S is $S^i \cup \{a_i\}$, where a_i is the lexicographically largest attribute that is not contained in S.*

Proof. Let $S_1 = S^i \cup \{a_i\}$, with a_i being the lexicographically largest attribute that is not contained in S. Suppose the lemma is not true, then there must exist S_2, such that $S \ll S_2 \ll S_1$. Since $S \ll S_2$, there must exist an attribute $a_j (i \neq j)$, which satisfies $a_j \in S_2$, $a_j \notin S$ and $S^{j-1} = S_2^{j-1}$. We also know that a_i is the smallest attribute that differentiates S and S_1, so $S^{i-1} = S_1^{i-1}$. We consider the following two possible relationships between a_i and a_j.

$a_i <_L a_j$: Since $i<j$, $S \ll_j S_2$ implies a_j is not contained in S, which contradicts the fact that a_i is the largest attribute not contained in S.
$a_i >_L a_j$: Since $i>j$, $S^{i-1} = S_1^{i-1}$ implies $S^{j-1} = S_1^{j-1}$. And we also have $S^{j-1} = S_2^{j-1}$, so $S_1^{j-1} = S_2^{j-1}$. Since the smallest attribute that differentiates S and S_1 is a_i, which is larger than a_j, so $a_j \notin S_1$. Since $S_1^{j-1} = S_2^{j-1}$, $a_j \in S_2$ and $a_j \notin S_1$, we have $S_1 \ll S_2$, which contradicts the assumption $S \ll S_2 \ll S_1$. □

Starting from the empty subspace, if we keep looking for the next lectically smallest subspace, we actually perform a right-to-left pre-order depth-first

traversal of the prefix tree. For the example shown in Fig. 1, the total lectical order is: $\{\phi\} \ll_d \{d\} \ll_c \{c\} \ll_d \{cd\} \ll_b \{b\} \ll_d \{bd\} \ll_c \{bc\} \ll_d \{bcd\} \ll_a \{a\} \ll_d \{ad\} \ll_c \{ac\} \ll_d \{acd\} \ll_b \{ab\} \ll_d \{abd\} \ll_c \{abc\} \ll_d \{abcd\}$.

The next question is how to find the next closed subspace after S. Let a_i be the lexicographically largest attribute that is not contained in S. If $S \cup \{a_i\}$ is a closed subspace, then it is trivial that $S \cup \{a_i\}$ is the next closed subspace. If $S \cup \{a_i\}$ is not closed, then its closure $\varphi \circ \psi(S^i \cup \{a_i\})$ must contain an attribute $a_j <_L a_i$ and $a_j \notin S$. To simplify the notation, we define $\mathbf{S} \oplus \mathbf{a_i} := \varphi \circ \psi(\mathbf{S^i} \cup \{\mathbf{a_i}\})$. Lemma 3 shows the method to find the next closed subspace after S.

Lemma 3. *The lectically smallest closed subspace that is lectically larger than S is $\varphi \circ \psi(S^i \cup \{a_i\})$, where a_i is the lexicographically largest attribute that is not contained in S for which $S \ll_i S \oplus a_i$ holds.*

A detailed proof for Lemma 3 can be found in [5]. Let a_i be the lexicographically largest attribute that is not contained in S for which $S \ll_i S \oplus a_i$ holds. Let a_k be an attribute $a_k \notin S$ and $a_k >_L a_i$. Since $S \not\ll_k S \oplus a_k$, $S \oplus a_k$ must contains at least one attribute that is lexicographically smaller than a_k. Let $S \ll_j S \oplus a_k$, that is, a_j is the lexicographically smallest attribute that differentiates S and $S \oplus a_k$. If $a_j <_L a_i$, then $S \oplus a_k$ is lectically larger than $S \oplus a_i$. If $a_j = a_i$, then $S \oplus a_i = S \oplus a_k$. If $a_j >_L a_i$, this contradicts the assumption that a_i is the lexicographically largest attribute that is not contained in S for which $S \ll_i S \oplus a_i$ holds. So in conclusion, Lemma 3 is true.

2.4 Density and Weighted Density

Notice that many nodes in Fig. 2 contain empty object set, which do not contribute to the clustering process. Furthermore, simply enumerating all maximal 1-complete regions is very time consuming. So we focus on finding those maximal 1-complete regions that contain at least a certain number of objects. Formally, we define the density of a single attribute a_i to be the ratio between the number of '1' entries in a_i and the total number of objects in the data, denoted as $dens(a_i)$. For the example shown in Table 2, $dens(d)$ is $\frac{5}{7}$ and $dens(a)$ is $\frac{3}{7}$. Similarly, the density of a subspace cluster is the ratio between the number of objects contained in it and the total number of objects in the data space. For example, the density of $<\{1,2,4,6,7\},\{cd\}>$ is $\frac{5}{7}$.

The weighted density of a subspace cluster $C = <O, A>$, denoted as $dens_w(C)$, is defined as the ratio between $dens(C)$ and the average density over all attributes contained in A, that is, $dens_w(C) = \frac{dens(C)}{\frac{1}{|A|}(\sum_{a_i \in A} dens(a_i))}$, where $|A|$ is the number of attributes contained in S. We call the denominator, $\frac{1}{|A|}(\sum_{a_i \in A} dens(a_i))$, the weight.

Since each subspace S has a unique closure $\varphi \circ \psi(S)$, which corresponds to exactly one maximal 1-complete region $C = <\psi(S), \varphi \circ \psi(S)>$, we define the density of subspace S ($dens(S)$) to be $dens(C)$, where C is the cluster having

the closure of S ($\varphi \circ \psi(S)$) as its attribute set. Similarly, $dens_w(S)$ is equal to $dens_w(C)$.

The next section presents the algorithm for finding all maximal 1-complete regions that have a weighted density larger than δ, where δ is a real number between 0 and 1.

3 Mining Weighted Dense Maximal 1-complete Regions

In this section, we present the underlying idea of our algorithm and the proof of correctness. Then we present some methods to speed up the algorithm.

3.1 Non-Decreasing Property

As shown in Fig. 2, density is non-increasing along any branches in the tree. This is because that the set of objects that are contained in a child node $S \cup \{a_i\}$ is the intersection of $\psi(\{a_i\})$ and the object set of its parent node S. Consequently, $\psi(S \cup \{a_i\})$ must be a subset of $\psi(S)$.

However, weighted density does not have this property. Although the density is non-increasing (numerator), the weights (denominator) may decrease when less dense attributes are added. If the decrease of the weights is faster than the decrease of density, weighted density of a child node may become larger than its parent node. One way to guarantee that weighted density is non-increasing along any branches is to enforce a constraint on the lexicographical order. More specifically, we sort all the attributes into the increasing density order, such that the lexicographically largest attribute is the one that has the largest density. By doing this, we can make sure that when we go deeper into the tree, the weights never decrease. Therefore, weighted density along any branches of the tree must also be non-increasing. This property facilitates the search algorithm that is introduced later.

In the remaining of the chapter, we assume that the data has been sorted this way. For the data shown in Table 2, the sorted dataset is shown in Table 3.

Table 3. Sorted example

	a	b	c	d
1	0	0	1	1
2	1	0	1	1
3	1	1	0	1
4	0	0	1	1
5	1	1	0	0
6	0	0	1	1
7	0	0	1	1
8	0	1	0	0

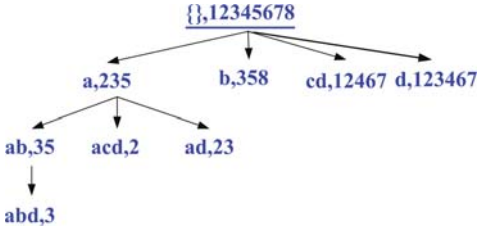

Fig. 3. 1-complete regions for sorted data

Figure 3 is a tree containing all and only the maximal 1-complete regions in this sorted dataset. Ideally, we only need to check all and only the nodes in Fig. 3, which is much smaller than the number of nodes contained in the complete tree as shown in Fig. 1.

3.2 Mining Weighted Dense 1-complete Regions

To better explain the algorithm, we first show the underlying idea and the correctness proof of our approach. Lemma 4 states that under certain condition, applying the "$\oplus a_i$" operator multiple times has the same effect as applying only once.

Lemma 4. $S \ll_i S \oplus a_i \rightarrow S \oplus a_i \oplus a_i = S \oplus a_i$.

Rationale. By definition, $S \ll_i S \oplus i$ means that $S \cap \{a_1, a_2, \ldots, a_{i-1}\} \cup \{a_i\} = S \oplus a_i \cap \{a_1, a_2, \ldots, a_{i-1}\} \cup \{a_i\}$. Since $S \oplus a_i \oplus a_i = \varphi \circ \psi(S \oplus a_i \cap \{a_1, a_2, \ldots, a_{i-1}\} \cup \{a_i\})$, and $\varphi \circ \psi(S \cap \{a_1, a_2, \ldots, a_{i-1}\} \cup \{a_i\}) = S \oplus a_i$, we have $S \oplus a_i \oplus a_i = S \oplus a_i$.

Lemma 5. $S \ll_i S \oplus a_i$ *and* $a_j >_L a_i \rightarrow S \oplus a_i \oplus a_j \supset S \oplus a_i$.

Rationale. $S \oplus a_i \oplus a_j = \varphi \circ \psi(S \oplus a_i \cap \{a_1, a_2, \ldots, a_{j-1}\} \cup \{a_j\})$. Since $a_i \in S \oplus a_i$ and $a_j >_L a_i$, $S \oplus a_i \cap \{a_1, a_2, \ldots, a_{j-1}\} \cup \{a_j\} \supset S \oplus a_i \cap \{a_1, a_2, \ldots, a_{i-1}\} \cup \{a_i\}$. This implies $\varphi \circ \psi(S \oplus a_i \cap \{a_1, a_2, \ldots, a_{j-1}\} \cup \{a_j\}) \supset \varphi \circ \psi(S \oplus a_i \cap \{a_1, a_2, \ldots, a_{i-1}\} \cup \{a_i\})$, which is equivalent to $S \oplus a_i \oplus a_j \supset S \oplus a_i$.

The implication of Lemma 5 is that if a 1-complete region C_1 in subspace $S \oplus a_i$ does not have enough density, then there is no need to check any attribute $a_j >_L a_i$. This is because Lemma 5 proves that $S \oplus a_i \oplus a_j$ is a superset of $S \oplus a_i$, thus the cluster C_2 in $S \oplus a_i \oplus a_j$ must have a density less than $dens(C_1)$. Furthermore, since the weights is non-decreasing along any branches after we sort the attributes into increasing density, $dens_w(C_2)$ must also be less than $dens_w(C_1)$. Thus if we know that $dens_w(C_1) < \delta$, $S \oplus a_i \oplus a_j$ can be safely pruned. Similarly, we can prove the following Lemma 6 by induction.

Lemma 6. $S \ll_i S \oplus a_i$ *and* $a_{k_m} >_L \ldots >_L a_{k_2} >_L a_{k_1} >_L a_i \rightarrow S \oplus a_i \oplus a_{k_1} \ldots \oplus a_{k_m} \supset S \oplus a_i$.

Lemma 6 tells us that if a 1-complete region C_1 in subspace $S \oplus a_i$ does not have enough weighted density, we can directly jump to test $S \oplus a_j$ for $a_j <_L a_i$ because anything in between must not meet the minimum weighted density threshold, which leads to Theorem 1.

Theorem 1. *The lectical smallest closed subspace larger than a given subspace $S \subset \mathcal{A}$ and having weighted density larger than δ is $S \oplus a_i$, where a_i is the lexicographically largest attribute which satisfies $dens_w(S \oplus a_i) > \delta$ and $S \ll_i S \oplus a_i$.*

Rationale. Let $S \oplus a_j$ be the lectically smallest closed subspace that is larger than S. If $dens_w(S \oplus a_j) > \delta$, the theorem is true since it is the same case as in Lemma 3. If $dens_w(S \oplus a_j) < \delta$, let a_i be the largest attribute for which $a_i <_L a_j$ and $S \ll_i S \oplus a_i$ hold. So we need to show that $S \oplus a_j \oplus a_i$ is the lectically smallest closed subspace that is larger than $S \oplus a_j$, and potentially could have enough weighted density. Since $dens_w(S \oplus a_j) < \delta$, Lemma 6 guarantees the search to start with a_{j-1} for the smallest weighted dense cluster. Since $S \ll_j S \oplus a_j$, $S \cap \{a_1, \ldots, a_{j-1}\} = S \oplus a_j \cap \{a_1, \ldots, a_{j-1}\}$. So the search for the next a_i performs the same on S and $S \oplus a_j$, that is, $S \oplus a_i = S \oplus a_j \oplus a_i$. So $S \oplus a_i$ is the lectically smallest closed subspace that is larger than S and could have enough weighted density. If $dens(S \oplus a_i) > \delta$, this theorem is true. Otherwise, find the next $a_k <_L a_i$ for which $S \ll_k S \oplus a_k$, and the proof can be completed inductively.

3.3 Lectical Weighted Dense Region Mining Algorithm

Theorem 1 states that if we find that a subspace $S \oplus a_i$ is not weighted dense, we can prune the search space by skipping all $a_j >_L a_i$, and check directly on a_{i-1} in the next iteration of the algorithm. Algorithm 1 is a straightforward implementation of this idea. Based on the correctness of Theorem 1, we can conclude the correctness of Theorem 2.

Algorithm 1 Lectical weighted dense region mining algorithm

1. $C = <O, S> \leftarrow <\psi(\phi), \varphi \circ \psi(\phi)>$
2. **IF** $(dens_w(C) > \delta)$
3. Add $C = <O, S>$ to Tree
4. $found \leftarrow true$
5. **END IF**
6. **REPEAT**
7. $(C,\ found) \leftarrow$ findnext(C)
8. **UNTIL** $found = false$

Theorem 2. *Algorithm 1 finds all maximal* 1*-complete regions that satisfy the minimum weighted density threshold δ.*

FUNCTION: findnext(C)
1. $found \leftarrow FALSE$
2. $o \leftarrow lexicographlly - largest(\{i|a_i \notin S\})$
3. **WHILE** (!$found$) **AND** ($o \geq 0$)
4. **IF** ($a_o \notin S$)
5. $\overline{C} = <\overline{O}, \overline{S}> \leftarrow <\psi(S \oplus a_o), S \oplus a_o>$
6. **IF** ($dens_w(\overline{C})>\delta$) **AND** ($S \ll_o \overline{S}$)
7. $found \leftarrow TRUE$
8. Add $\overline{C} = <(\overline{O}), \overline{S}>$ to Tree
9. **END IF**
10. **END IF**
11. $o \leftarrow o - 1$
12. **END WHILE**
13. **RETURN** ($\overline{C}$, found)

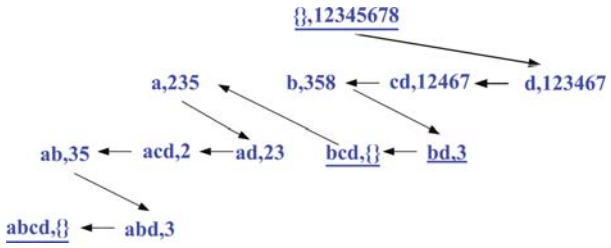

Fig. 4. Search tree of sorted data

The search starts out by finding the closure of the empty subspace (line 1), and adding that to the tree of closed subspace if it has enough weighted density (line 2–3). Then the algorithm keeps looking for the next lectically larger closed subspace satisfying the weighted density constraint until no more such subspaces can be found (line 6–8).

Function $findnextbasic$ accepts a 1-complete region C as parameter, and returns the next lectically smallest closed and weighted dense subspace and its corresponding maximal 1-complete region. First the flag $found$ is set to be false. Starting from the lexicographically largest attribute not contained in the current subspace S, it looks for an attribute a_o that meets the two conditions at line 6. The loop terminates either with a successful candidate or when all the possibilities have been tried (line 3).

Figure 4 traces the algorithm on the dataset shown in Table 3 with $\delta = 0$ (no weighted density pruning). Nodes in the tree are those being visited. Underlined nodes are non-maximal ones. The arrows indicate the sequence of visiting. Suppose we start from node $S = \phi$. Since the current subspace is empty, the largest attribute not contained in S is d. Then we compute the $S \oplus \{d\} = \{d\}$. Since $S \ll_d S \oplus \{d\}$, we output cluster $<\{123467\}, \{d\}>$ and keep looking for the next one.

3.4 Optimizing Techniques

In this section, we present several methods to optimize the time complexity of the basic algorithm. The data is stored as bit strings, that is, each attribute is represented as a string of 0 and 1. The major operation of our algorithm is bit intersection. When the percentage of 1 entries in the dataset is larger than 10%, using bit strings not only saves memory space, it also makes the computations more efficient.

Reuse Previous Results in Computing $\overline{O}$

The most expensive operation in Function $findnext$ is at line 5, where we need to compute the $\overline{S} = S \oplus a_o$ and its object set $\overline{O} = \psi(S \oplus a_o)$. Notice that for any node in the prefix tree as shown in Fig. 2, its object set can be computed incrementally from the object set of its parent. That is, the object of the child node is the intersection of the object set of the parent node and $\psi(a_o)$, where a_o is the newly added attribute. For example, the object set of cd can be computed by taking the intersection of the object set of its parent node $c(\{123467\})$ and $\psi(d)$ $(\{12467\})$. So we can maintain the object sets of all the nodes on the current branch of the search tree on a stack called $curPath$ to avoid duplicated intersection operations.

However, when the search moves from one branch to the other, the stack $curpath$ needs to be updated to maintain the correctness of the object set computation. For example, after we visited node ad, the next node to be visited is ac. But the object set of ac can not be incrementally computed based on the object set of ad, while it can be computed incrementally based on the object set of a. So we maintain another stack of attribute id called $istack$, which keeps track of all the attribute id for which $S \ll_o S \oplus a_o$ is true. For example, after we find that the next closed subspace after $\varphi \circ \psi(\phi)$ is $<\{123467\}, c>$, we push the object set into $curPath$ and we push c into stack $istack$. When we try to find the next closed subspace after c, we check if o is larger than the top of $istack$. If yes, that means that we are still on the same branch of the search tree, so there is no need to change the stack; if no, that means that we are jumping to a different branch, so pop up all the elements in $iStack$ that is larger than o. When popping out the elements in $iStack$, $curPath$ is also updated in the similar fashion. That is, whenever pop out an element from $iStack$, we also pop out an element from $curPath$.

Stack of Unpromising Nodes

Observe the search tree in Fig. 2. Starting from node ϕ, we first check if $\phi \ll_d \phi \oplus d$. Since $\phi \oplus d = \{cd\}$, we know that any closed subspaces that contain d must also contain c. So, after we reach node $\{a\}$, there is no need to check $\{ad\}$, since we know for sure that it can not be closed. For this type of pruning, we maintain a stack called $prelistStack$. This stack contains elements called

prelist, and for each attribute i, $prelist[i]$ is the id of the lexicographically smallest attribute j for which $\psi(j) \supseteq \psi(i)$. Initially set all $prelist[i] = i$. During the search algorithm, set the elements accordingly. Similar to $curPath$ and $iStack$, *prelist* needs to be updated when we jump between branches.

4 Experimental Results

We tested our algorithm on three datasets as listed in Table 4, which includes the name of the dataset, number of objects, number of attributes, minimum density of the attributes, and maximal density of the attributes. Mushroom and Chess are from [4], and Cog is from [10].[1] The objective of the experiments is to show that our algorithm can indeed find clusters both from dense subspaces and relatively sparse subspaces. All our experiments were performed on 2.4 GHz Pentium PC with 512 MB memory running Windows 2000.

All test data are very dense in the sense that the number of maximal 1-complete regions contained in the datasets is much larger than the number of objects in the datasets. Another feature of these data is that their attributes have quite different densities. Mushroom contains 129 attributes and 8,124 objects, while the most dense attribute contains all '1's and the least dense attribute contains only four '1's. The other two datasets have similar characteristics. Figure 5 shows the density distribution of the attributes for all the three datasets. For the Chess dataset, around 30% of the attributes have density less than 20%. If we set the minimum density to be 20%, we will not be able to find any patterns in almost one thirds of the subspaces. One possible solution to find patterns in these less dense subspaces is to reduce the minimum density threshold to less than 20%. However, reducing the minimum density threshold leads to an exponential growth in the total number of clusters being found, most of which belong to the more dense subspaces. So we perform the following experiments to show that our algorithm can find weighted dense 1-complete regions in both dense subspaces and sparse subspaces.

We compared our algorithm with $CLOSET+$ [11], which is an enhanced version of $CLOSET$ [9]. For $CLOSET+$, a very small minimum density threshold value is needed in order to find those weighted dense clusters in the less dense subspaces. We set the minimum density threshold for $CLOSET+$ to be a value such that it can find all weighted dense regions larger than a

Table 4. Datasets characteristics

	# Of objs	# Of attrs	Minimum density	Maximum density
Mushroom	8,124	129	0.01	1
Chess	3,196	75	0.03	1
COG	3,307	43	0.11	0.60

[1] Cog stands for clusters of orthologous genes.

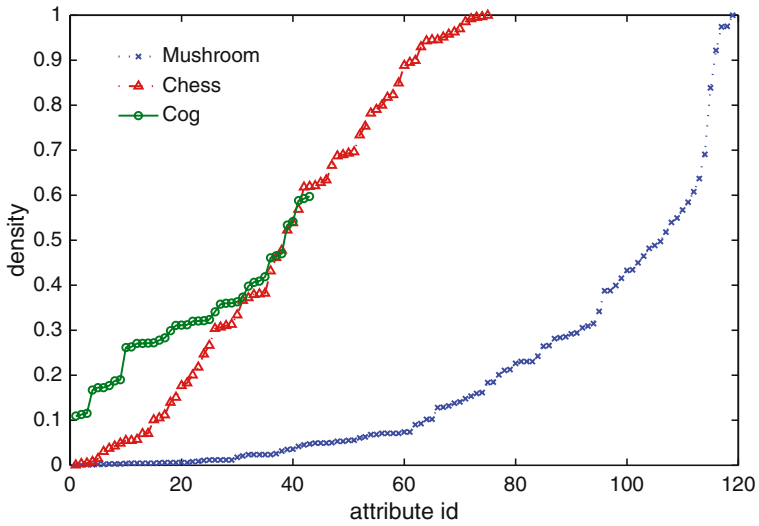

Fig. 5. Density distribution for all attributes

certain threshold value. For example, the least dense attribute in COG has density 0.11. If we want to find weighted dense clusters that have this least dense attribute, the minimum density threshold must be set to be no larger than 0.11. However, our tests show that for Chess and COG, $CLOSET+$ runs out of memory for these low threshold values. For Mushroom, $CLOSET+$ can finish the mining task for all threshold values.

Our algorithm uses almost the same amount of memory for all weighted density threshold values, since the computation of the next cluster depends only on the current cluster and not on any other previously found ones. As shown in Fig. 6, our algorithm uses almost the same amount of memory for all weighted density threshold values for all datasets. Compared with $CLOSET+$, our algorithms uses much less memory on Mushroom. For Chess and COG, the difference is more significant as $CLOSET+$ cannot finish the task due to insufficient memory.

We also compared the running time of our algorithm with $CLOSET+$ on the Mushroom data. Since $CLOSET+$ runs out of memory on Chess and Cog, we only report the running time for our algorithm. In order to find weighted dense clusters in the least dense subspaces, $CLOSET+$ needs to find almost all dense regions, which explains why its running time is almost constant for all threshold values. Even if we want to find all the maximal 1-complete regions in the data, our algorithm is still faster than $CLOSET+$.

Figure 9 shows the total number of clusters being found for various weighted density threshold values. For all three datasets, the running time curves as shown in Figs. 7 and 8 fit very well with the curves in Fig. 9. This suggests that our algorithm has a linear time complexity with the number of clusters being found.

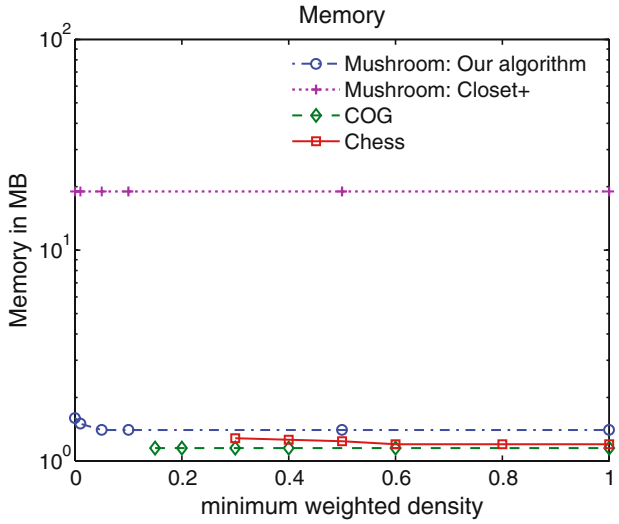

Fig. 6. Memory comparison

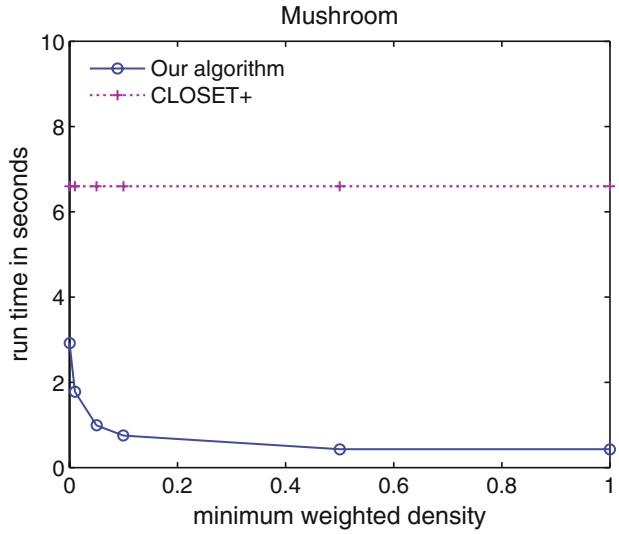

Fig. 7. Mushroom time comparison

We also want to show through experiments that using weighted density can find more clusters in less dense subspaces. So we compared the results from density pruning with the results from weighted density pruning. For fair comparison, we only compare when the minimum density threshold and the minimum weighted density threshold are equally selective, that is, there are equal number of clusters that satisfy each of the constraint. Figure 10 shows the percentage of the clusters being found after each attribute id on COG

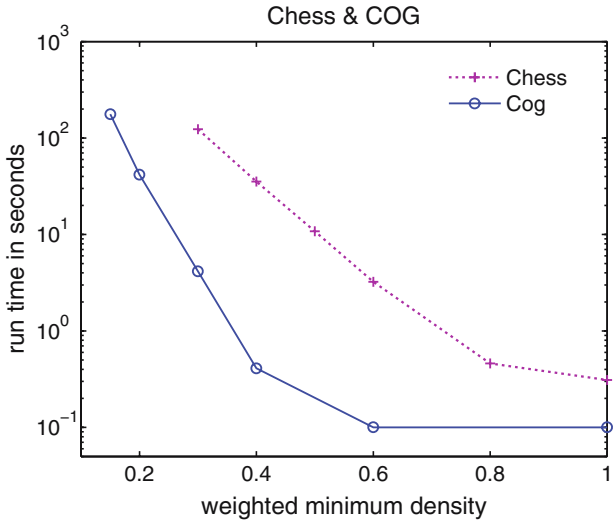

Fig. 8. Chess and COG run time comparison

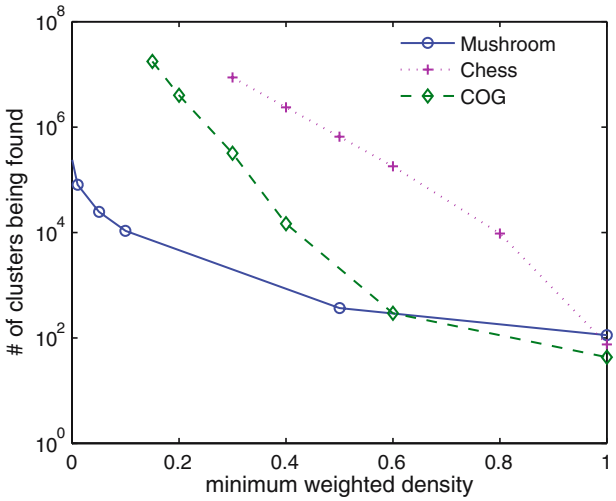

Fig. 9. Total number of clusters being found

when there are 10,000 clusters being found. Attributes are numbered such that more dense larger attributes have larger ids. The search starts from the attribute that has the largest id (45 in this case), and ends when it finishes attribute 0. From the figure we can see that when using weighted density, more clusters in the less dense subspaces are returned. Close examination reveals that using minimum density threshold, seven attributes are not included in any clusters. On the other hand, using weighted density, all attributes belong

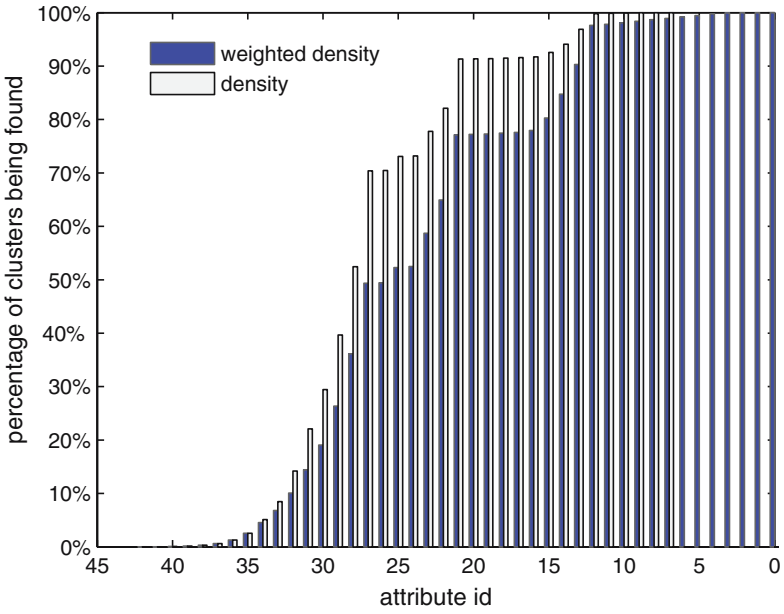

Fig. 10. Comparison between density and weighted density

to at least one cluster. We tested a set of different selective threshold values on all three datasets, and all of them confirms that using the weighted density constraint finds more clusters in less dense subspaces.

5 Conclusion

We have presented a new subspace clustering mining algorithm to find weighted dense maximal 1-complete regions in high dimensional datasets. Our algorithm is very memory efficient, since it does not need to keep all the clusters found so far in the memory. Unlike other density mining algorithms which tend to find only patterns in the dense subspaces while ignore patterns in less dense subspaces, our algorithm finds clusters in subspaces of all densities. Our experiments showed that our algorithm is more efficient than $CLOSET+$ from both time complexity and memory consumption perspectives.

References

1. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications, *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*, ACM, New York, June 1998, 94–105

2. Agrawal, R., Imielinski, T., Swami, A.: Mining Association Rules Between Sets of Items in Large Databases, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of data (SIGMOD'93)*, ACM, New York, May 1993, 207–216
3. Agrawal, R., Srikant, R.: *Fast Algorithms for Mining Association Rules*, Morgan Kaufmann, Los Altos, CA, 1998, 580–592
4. Blake, C., Merz, C.: UCI Repository of machine learning databases, http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998
5. Ganter, B., Kuznetsov, S.O.: Stepwise Construction of the Dedekind–MacNeille Completion, *Proceedings of Sixth International Conference on Conceptual Structures (ICCS'98)*, August 1998, 295–302
6. Ganter, B., Wille, R.: *Formal Concept Analysis: Mathematical Foundations*, Springer, Berlin Heidelberg New York, 1999
7. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules, *Proceeding of the Seventh International Conference on Database Theory (ICDT'99)*, Springer, Berlin Heidelberg New York, January 1999, 398–416
8. Peeters, R.: The maximum edge biclique problem is NP-complete, *Discrete Applied Mathematics*, **131**, 2003, 651–654
9. Pei, J., Han, J., Mao, R.: CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets, *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, May 2000, 21–30
10. Roman, T., Natalie, F., et al.: The COG database: an updated version includes eukaryotes, *BMC Bioinformatics*, **4**, September 2003
11. Wang, J., Han, J., Pei, J.: CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets, *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, ACM, New York, 2003, 236–245
12. Zaki, M.J., Hsiao, C.J.: Charm: an Efficient Algorithm for Closed Itemset Mining, *Proceedings of the Second SIAM International Conference on Data Mining (SDM'04)*, April 2002