A Linear Delay Algorithm for Building Concept Lattices

Martin Farach-Colton and Yang Huang [⋆]

Department of Computer Science, Rutgers University, Piscataway, NJ 08854,
farach@cs.rutgers.edu yahuang@cs.rutgers.edu

Abstract. Concept lattices (also called Galois lattices) have been applied in numerous areas, and several algorithms have been proposed to construct them. Generally, the input for lattice construction algorithms is a binary matrix with size $|G||M|$ representing binary relation $I \subseteq G \times M$. In this paper, we consider polynomial delay algorithms for building concept lattices. Although the concept lattice may be of exponential size, there exist polynomial delay algorithms for building them. The current best delay-time complexity is $O(|G||M|^2)$. In this paper, we introduce the notion of *irregular concepts*, the combinatorial structure of which allows us to develop a linear delay lattice construction algorithm, that is, we give an algorithm with delay time of $O(|G||M|)$. Our algorithm avoids the union operation for the attribute set and does not require checking if new concepts are already generated. In addition, we propose a compact representation for concept lattices and a corresponding construction algorithm. Although we are not guaranteed to achieve optimal compression, the compact representation can save significant storage space compared to the full representation normally used for concept lattices.

1 Introduction

Concept lattices have proved useful in many areas, such as knowledge representation [13], information retrieval [3], web document management [4, 8], software engineering [16] and bioinformatics [6, 11]. Of particular importance in these applications is the structure of the lattice, i.e. the *Hasse diagram* of the concept lattices. For example, the immediate predecessors and successors of a concept are used in browsing web documents [8] or to infer the class hierarchy of a program [16]. The edge information of Hasse diagrams can be used to compare two concept lattices in which gene expression information has been coded [6].

A concept lattice can be briefly defined as follows. Given a binary relation between an object set and an attribute set, a *concept* is a pair of object set A and attribute set B, denoted as (A, B), where A contains all objects sharing every attribute in B and B contains all attributes shared by every object in A. A concept lattice is the partially ordered set of all concepts, in which the order

[⋆] Yang Huang is currently at NCBI/NLM/NIH, 8600 Rockville Pike, Room 8N811I, Bethesda, MD 20894. This work was done when he was at Rutgers University.

is defined using subset order on object sets (which is equivalent to containment order on attribute sets).

In real-world applications, we often find that it is necessary to construct a concept lattice from large amount of input data. It is known that the number of concepts in a concept lattice can be exponential in the size of the input binary matrix. The problem of deciding the number of concepts has been shown to be #P-complete [19].

Constructing a concept lattice is a type of *enumeration problem*. Algorithms for enumeration problems are typically measured both by *total time complexity* and *delay-time complexity* [12]. The running time of algorithms with polynomial total time is a polynomial in the size of the input and output. Polynomial delay time means that there is a polynomial in the input size that bounds the time to the first entity outputted as well as the delay between any two consecutive output entities. An algorithm with polynomial delay-time complexity has total polynomial running time, which is the polynomial delay time multiplied by the output size. Algorithms with polynomial delay time are often preferred because they allow us to predict the time to get the next entity. They allow the procedure of entity processing to follow immediately. They also allow to generate a subset of entities without generating others.

Related work. The problem of generating the set of concepts is closely related to two other important problems: generating all maximal bipartite cliques in a given bipartite graph $\mathcal{G}_b = (V_1, V_2, E)$ and generating all frequent closed itemsets in a transaction database [21]. Note that generating all maximal bipartite cliques or all frequent closed itemsets is not enough for generating a concept lattice since a concept lattice requires the partial order among all concepts be recovered. A maximal bipartite clique corresponds to a concept if we consider $\mathcal{G}_b$ as a representation for the matching relation between two parts of $\mathcal{G}_b$'s vertices. A frequent closed itemset corresponds to a concept whose object set size is larger than a certain threshold. Eppstein [7] showed that the number of all maximal bipartite cliques is $O(|V(\mathcal{G})|)$ in a graph with bounded arboricity and gave a linear total time algorithm, where $V(\mathcal{G})$ is the vertex set of $\mathcal{G}$. An algorithm for generating all maximal bipartite cliques in any bipartite graph was designed by Makino and Uno [14]. It takes $O(\Delta^2)$ polynomial delay, where Δ is the maximum degree of $\mathcal{G}_b$. Given $|V_1| = |G|$ and $|V_2| = |M|$, then $\Delta = \max(|G|, |M|)$. CLOSET+ [18], CHARM [20] and LCM2 [17] are among state-of-the-art algorithms for generating the set of frequent closed itemsets.

However, though many algorithms are available for generating the set of concepts and some of them are quite fast, few algorithms compute the edge structure of the lattice. Bordat's algorithm [2] uses a trie to store and retrieve concepts with delay-time complexity $O(|G||M|^2)$, where G is the input object set and M is the input attribute set. Without loss of generality, we will assume $|G| \geq |M|$. Depending on the value of Δ, the delay-time complexity of Makino and Uno's algorithm [14] may be better than $O(|G||M|)$. However, it can not be bound by $O(|G||M|)$ in the worst case. The best polynomial delay-time complexity of algorithms for constructing a concept lattice in terms of $|G|$ and $|M|$ is $O(|G||M|^2)$.

Godin *et al.* [10] proposed an incremental algorithm that dynamically updates the structure of the concept lattice as new rows or columns are added to the input matrix. The algorithm by Nourine and Raynaud [15] has the best known total time complexity $O(|G||M||\mathcal{B}|)$, where $\mathcal{B}$ is the set of all concepts. But it is not a polynomial delay algorithm. Recently, Choi [5] proposed an efficient concept lattice construction algorithm with complexity $O(\sum_{a \in ext(\mathcal{C})} |cnbr(a)|)$, where $ext(\mathcal{C})$ is the object set of the concept $\mathcal{C}$ and $cnbr(a)$ is a reduced attribute set of a. However, it seems to us that the condition used in the algorithm, which is to check if a newly generated pair of object set and attribute set is a concept, is not sufficient. Berry *et al.* [1] suggested constructing concept lattices by searching non-dominating maxmods in a co-bipartite graph. The complexity is $O(|G||M|)$ per concept plus $O(|G||M|^2)$ per traversed maximal chain of the lattice.

Our results. In this paper, we propose a concept lattice construction algorithm with delay $O(|G||M|)$, which is linear in the size of the input matrix. Though the total time complexity of our algorithm, $O(|G||M||\mathcal{B}|)$, ties with that of Nourine and Raynaud, our algorithm is a polynomial delay algorithm, which their algorithm is not. By introducing the set of irregular concepts, we ensure that when we compute the union of several attribute sets they are disjoint. Our algorithm also avoids the operation to check if a newly generated pair of object set and attribute set is a concept or if it is going to be subsumed, as most previous algorithms do.

The usual way to represent a concept is by a pair of its object set and attribute set, which contain a lot of redundant information. We call this the *full representation*. The space required for storing the full representation of all concepts is $O(|G||\mathcal{B}|)$. We propose a *compact representation* for concept lattices, in which we represent a concept in terms of the set difference between its object/attribute set and the one in one of its predecessors. In the optimal case, a compact representation only requires $O(|\mathcal{B}|)$ space for all concepts, which reaches the lower bound. Given the compact representation of a concept lattice, we can easily recover the full representation in linear time. We modify our algorithm for the full representation to construct a compact representation.

From now on, we will refer to concept lattices as lattices from time to time when the context is clear. The remaining of the paper is organized as follows: We introduce the basics of lattices in section 2. In section 3 we present some characterization for lattices. We introduce our algorithm for the full representation in section 4 and the modified version for the compact representation in section 5. Finally, we conclude in section 6 and discuss some future research direction.

2 Preliminary

In this section we will give a brief overview for lattices. For a complete introduction, please refer to the book [9]. Many of our notations follow the ones used in the book. Given a context (G, M, I) where G is the *object* set and M is the *attribute* set, a binary matrix R is used to represent the relation $I \subseteq G \times M$, i.e. $R_{i,j} = 1$ if $(g_i, m_j) \in I$ where $g_i \in G$ and $m_j \in M$ and $R_{i,j} = 0$ otherwise. For $g_i \in G$, we define $g_i' = \{m_j | R_{i,j} = 1\}$. Furthermore, for an object set $A \subseteq G$, we denote $A' = \cap_{g_i \in A} g_i'$. Dually, we define $m_j' = \{g_i | R_{i,j} = 1\}$ for $m_j \in M$ and $B' = \cap_{m_j \in B} m_j'$ for $B \subseteq M$. With the above notation we are ready to define the concept.

Definition 2.1. *The* concept *is a pair (A, B) where $A \subseteq G$, $B \subseteq M$, $A = B'$ and $B = A'$. A is called* extent *and B is called* intent *of the concept.*

For a concept $\mathcal{C} = (A, B)$, we denote A as $ext(\mathcal{C})$ and B as $int(\mathcal{C})$. We call a set A *closed* if $A = A''$. The extent and intent of a concept are closed sets. It can be seen that a closed object set A or a closed attribute set B uniquely determines a concept (A, A') or (B', B).

A partial order $\preceq$ is defined on $\mathcal{B}$, the set of all concepts:

Definition 2.2. *If $ext(\mathcal{C}) \subseteq ext(\mathcal{D})$ ($int(\mathcal{D}) \subseteq int(\mathcal{C})$), then $\mathcal{C} \preceq \mathcal{D}$. $\mathcal{C}$ is called* successor *of $\mathcal{D}$ and $\mathcal{D}$ is called* predecessor *of $\mathcal{C}$.*

Please note that definition of predecessors and successors in a concept lattice may be somewhat counter-intuitive. However, this way successors will be placed in a lower lever, below its predecessors, in the diagram representing a concept lattice. The diagram will be shown next. According to the definition, a concept is a predecessor and a successor of itself. In particular, if $\mathcal{C}$ is a successor of $\mathcal{D}$ other than $\mathcal{D}$ itself and $\forall \mathcal{E}$ such that $\mathcal{C} \preceq \mathcal{E} \preceq \mathcal{D}$ implies $\mathcal{E} = \mathcal{C}$ or $\mathcal{E} = \mathcal{D}$, then $\mathcal{C}$ is an *immediate successor* of $\mathcal{D}$ and $\mathcal{D}$ is an *immediate predecessor* of $\mathcal{C}$.

Definition 2.3. *The partially ordered set $\mathcal{L}(G, M, I) = \langle \mathcal{B}, \preceq \rangle$ is called* concept lattice *or* Galois lattice.

The diagram representing a partially ordered set is called *Hasse diagram*, where a vertex represents a concept, and two concepts are connected by an edge if one concept is an immediate successor of the other. We show a lattice example in the Figure 1.

Later we will need the definition of the infimum.

Definition 2.4. *The infimum of a subset S of a partially ordered set $(P, \preceq)$, denoted as $\wedge S$, is an element l of P such that*

1. *$\forall x \in S$, $l \preceq x$, and*
2. *for any $p \in P$ such that $\forall x \in S$, $p \preceq x$, it holds that $p \preceq l$.*

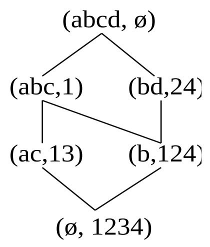
3 Some Characterization of Lattices

In this section, we will present some characterization of lattices, which will help us design the lattice construction algorithm. Due to the limit of space, all the proof is omitted.

We need the following known result:

	1	2	3	4
a	×		×	
b	×	×		×
c	×		×	
d		×		×

(a)



(b)

Fig. 1. (a) A binary matrix representing I, where the entry corresponding to g_i and m_j is x iff $(g_i, m_j) \in I$. $G = \{a, b, c, d\}$ and $M = \{1, 2, 3, 4\}$. (b) The Hasse diagram of the lattice constructed from (G, M, I). The lattice is represented in full representation.

Proposition 3.1. *[9] For a concept $\mathcal{C} \in \mathcal{B}$ in the lattice $\mathcal{L}$, $\mathcal{C}$ and all of its successors forms a concept lattice, denoted by $\mathcal{L}^{\mathcal{C}}$.*

In the following, we will first define regular and irregular concepts. Then we will present a lemma on the concept $\mathcal{C}$ and its irregular successors.

Suppose $\mathcal{D}$ is a concept in the lattice $\mathcal{L}^{\mathcal{C}}$, and denote the set of its immediate predecessors that are successors of $\mathcal{C}$ by $IP^{\mathcal{C}}_{\mathcal{D}}$. In addition, suppose $IP^{\mathcal{C}}_{\mathcal{D}} = \{\mathcal{D}_i | i \in [1..n]\}$, we denote the set $\bigcup_{i=1}^{n} int(\mathcal{D}_i)$, the union of the intent of concepts in $IP^{\mathcal{C}}_{\mathcal{D}}$, by $int(IP^{\mathcal{C}}_{\mathcal{D}})$.

Definition 3.1. *If $int(\mathcal{D}) = int(IP^{\mathcal{C}}_{\mathcal{D}})$, $\mathcal{D}$ is called* regular *concept of $\mathcal{C}$. If $int(\mathcal{D}) \supset int(IP^{\mathcal{C}}_{\mathcal{D}})$, $\mathcal{D}$ is called* irregular *concept of $\mathcal{C}$.*

Note that an irregular concept $\mathcal{D}$ of $\mathcal{C}$ is not necessarily *meet-irreducible* in $\mathcal{L}^{\mathcal{C}}$, where $\mathcal{D}$ is meet-irreducible if $\mathcal{D} = \wedge\{\mathcal{E}, \mathcal{F}\} \Rightarrow \mathcal{D} = \mathcal{E}$ or $\mathcal{D} = \mathcal{F}$, because $\mathcal{D}$ can have more than one immediate predecessor in $\mathcal{L}^{\mathcal{C}}$.

Any immediate successor of $\mathcal{C}$ is an irregular concept of $\mathcal{C}$. We denote the set of all $\mathcal{C}$'s irregular concepts by $IR^{\mathcal{C}}$. The following proposition will help us identify immediate successors of $\mathcal{C}$ from $IR^{\mathcal{C}}$.

Proposition 3.2. *Given $\mathcal{C}_i \in IR^{\mathcal{C}}$, it is an immediate successor of $\mathcal{C}$ if and only if there is no $\mathcal{C}_j \in IR^{\mathcal{C}}, j \neq i$ such that $\mathcal{C}_i \preceq \mathcal{C}_j$.*

The following lemma shows one of important properties of $IR^{\mathcal{C}}$:

Lemma 3.1. *Suppose $IR^{\mathcal{C}} = \{\mathcal{C}_i | i \in T\}$, where T is an index set. Given $\mathcal{C}_i \in IR^{\mathcal{C}}$, let $V_i = int(IP^{\mathcal{C}}_{\mathcal{C}_i})$ and let $B_i = int(\mathcal{C}_i) \setminus V_i$. If $i, j \in T$ and $i \neq j$, then $B_i \cap B_j = \emptyset$, and $\bigcup_{i \in T} B_i = \bigcup_{g \in ext(\mathcal{C})} g' \setminus int(\mathcal{C})$.*

The above lemma indicates that $\{B_i | i \in T\}$ constitutes a partition of $\bigcup_{g \in ext(\mathcal{C})} g' \setminus int(\mathcal{C})$, which is the set of attributes belonged to some $g \in ext(\mathcal{C})$ but not appearing in $int(\mathcal{C})$. Since $B_i \cap B_j = \emptyset$, $B_i \neq \emptyset, B_j \neq \emptyset$ and $B_i \subseteq M, B_j \subseteq M$, we have a direct corollary from the lemma:

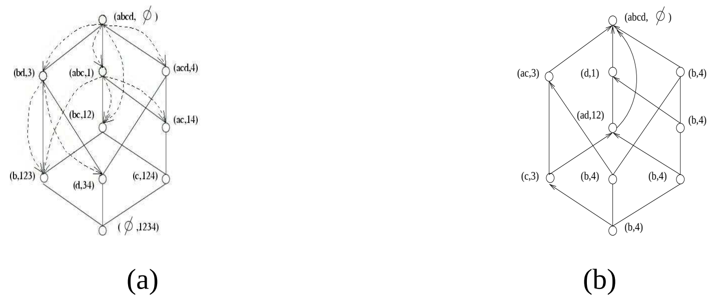


Fig. 2. (a) Irregular concepts regarding $(abcd, \emptyset)$, $(bd, 3)$ and $(abc, 1)$. The concepts pointed by an arrow and connected to $(abcd, \emptyset)$ $((bd, 3)/(abc, 1))$ by dashed lines are its irregular concepts. Note that the lattice is in the full representation. (b) The same lattice in a compact representation computed by our algorithm. For each concept, a directed edge points to its base.

Corollary 3.1. *For any concept $\mathcal{C}$, $|IR^{\mathcal{C}}| \leq |M|$.*

Since B_i is still "partial" compared to $int(\mathcal{C}_i)$, let us introduce the set $\{(ext(\mathcal{C}_i), B_i) | \mathcal{C}_i \in IR^{\mathcal{C}}\}$ as $PIR^{\mathcal{C}}$, where B_i is defined as in Lemma 3.1. Please note that the only difference between $PIR^{\mathcal{C}}$ and $IR^{\mathcal{C}}$ is that the attribute set in $PIR^{\mathcal{C}}$ is not complete yet. To see some examples of PIR, let $\mathcal{C} = (abcd, \emptyset)$, $\mathcal{C}_1 = (bd, 3)$ and $\mathcal{C}_2 = (abc, 1)$ in Figure 2 (a). Then $PIR^{\mathcal{C}} = \{(bd, 3), (abc, 1), (acd, 4), (bc, 2)\}$. $PIR^{\mathcal{C}_1} = \{(b, 1), (d, 4)\}$. And $PIR^{\mathcal{C}_2} = \{(b, 3), (bc, 2), (ac, 4)\}$.

For a concept in $\mathcal{L}^{\mathcal{C}}$, the following corollary makes it easy to identify its predecessors in $PIR^{\mathcal{C}}$.

Corollary 3.2. *Given $\mathcal{C}$, $\forall (A_i, B_i) \in PIR^{\mathcal{C}}$, $\forall \mathcal{D} \preceq \mathcal{C}$, if $int(\mathcal{D}) \cap B_i \neq \emptyset$, then $\mathcal{D} \preceq \mathcal{E}$, where $\mathcal{E} = (A_i, A_i')$.*

Since $IR^{\mathcal{C}}$ contains all $\mathcal{C}$'s immediate successors, we will be able to generate the sublattice $\mathcal{L}^{\mathcal{C}}$ if we can generate $IR^{\mathcal{C}}$. Actually we can obtain $IR^{\mathcal{C}}$ by augmenting the attribute set B_i in $PIR^{\mathcal{C}}$ in a certain way. The theorem that will be shown next provides the basis for the processing.

Given $\mathcal{C}_j \in IR^{\mathcal{C}} = \{\mathcal{C}_i | i \in T\}$, where T is an index set, we define an equivalence relation $\sim_j$ on the set $T \setminus \{j\}$. $i \sim_j k$ if $ext(\mathcal{C}_i) \cap ext(\mathcal{C}_j) = ext(\mathcal{C}_k) \cap ext(\mathcal{C}_j) \neq \emptyset$ for $i, k \in T \setminus \{j\}$. Let the resulting equivalence classes on T be $[j_1], [j_2], \ldots, [j_r]$. For each equivalence class $[j_h], h \in [1..r]$, let us denote $A_{j_h} = ext(\mathcal{C}_i) \cap ext(\mathcal{C}_j), i \in [j_h]$ and $B_{j_h} = \bigcup_{i \in [j_h]} B_i$.

When we proceed to our main theorem in this section, the following proposition will become useful:

Proposition 3.3. *$\{B_{j_h} | h \in [1..r]\}$ constitutes a partition of $\bigcup_{g \in ext(\mathcal{C}_j)} g' \setminus (int(\mathcal{C}) \cup B_j)$, where $[j_1], [j_2], \ldots, [j_r]$ are equivalent classes defined above.*

Theorem 3.1. *If $\mathcal{C}_j \in IR^{\mathcal{C}}$ is an immediate successor of $\mathcal{C}$, then $PIR^{\mathcal{C}_j} = \{(A_{j_h}, B_{j_h})|h \in [1..r]\}$.*

It is easy to extend the theorem to the case that $\mathcal{C}_j \in IR^{\mathcal{C}}$ is not an immediate successor of $\mathcal{C}$.

Corollary 3.3. *If $\mathcal{C}_j \in IR^{\mathcal{C}}$ is not an immediate successor of $\mathcal{C}$, then $PIR^{\mathcal{C}_j} \cup \{(ext(\mathcal{C}_j), int(\mathcal{C}_j) \setminus (int(\mathcal{C}) \cup B_j))\} = \{(A_{j_h}, B_{j_h})|h \in [1..r]\}$.*

4 Algorithm for the Full Representation

We will present a lattice construction algorithm which generates a lattice in the full representation. The input for the algorithm is a binary matrix R with $|G|$ rows and $|M|$ columns representing the relation I between the object set G and the attribute set M, where $R_{i,j} = 1$ if and only if $(g_i, m_j) \in I$.

4.1 Overview

The algorithm builds the lattice while traversing it in depth first search (DFS). Each node will represent a concept. Suppose the node $\mathcal{C}$, is already visited. And suppose $PIR^{\mathcal{C}}$ is already generated, each element of which is put in a child node of $\mathcal{C}$. These child nodes are sorted in the ascending order by the size of the object set in each child node. Though the intent of those concepts represented by the child nodes is not complete yet, we will still represent the nodes by $\mathcal{C}_j$. Note that the following conditions are met: Each shadow child node, which will be introduced below, of $\mathcal{C}$ contains (A_j, s_j) where $s_j = |B_j|, (A_j, B_j) \in PIR^{\mathcal{C}}$; All unvisited child nodes are of form $(A_j, B_j) \in PIR^{\mathcal{C}}$. When the algorithm visits a child node $\mathcal{C}_j = (A_j, B_j)$ for the first time, it begins to traverse the sublattice $\mathcal{L}^{\mathcal{C}_j}$. It will first generate $PIR^{\mathcal{C}_j}$ as $\mathcal{C}_j$'s child nodes, which contains all immediate successors of $\mathcal{C}_j$. To do so, the algorithm uses GeneratePIR($\mathcal{C}_j$, $\mathcal{C}$) and SearchEquiClass($\mathcal{C}_j$, $\mathcal{C}$) to generate $PIR^{\mathcal{C}_j}$ as Theorem 3.1 indicates. In GeneratePIR, we generate two kinds of child nodes using intersection on extents. One is marked as unvisited and the other is marked as *shadow*. There is no need to call GeneratePIR for a shadow node since its corresponding concept, say $\mathcal{F}$, and all $\mathcal{F}$'s successors have already been generated at that time because of DFS traversal. As we will see, a shadow node will never enter the stack. The shadow nodes are used to prevent generating a concept more than once without losing track of its immediate successors. Each shadow node has a pointer pointing to the corresponding concept in the lattice.

After the child nodes of $\mathcal{C}_j$ are generated, the algorithm uses SearchEquiClass to find equivalence classes $[j_r]$, where each class corresponds to $\mathcal{C}_j$ or a member in $IR^{\mathcal{C}_j}$. If a class contains a shadow node, it means that the concept corresponding to the class is already visited. If not, it means that the corresponding concept is unvisited yet. In both cases, we remove the child nodes under $\mathcal{C}_j$ for memory reuse. By Proposition 3.2, we check if there exists an equivalence class with $|A_{j_h}| = |A_j|$ to determine if $\mathcal{C}_j$ is an immediate successor of $\mathcal{C}$ or not. By

Proposition 3.3, when we generate B_{j_h} and $(A_j)'$ no actual union operation is required since B_is are disjoint. After constructing $\mathcal{L}^{\mathcal{C}_j}$ by repeatedly calling GeneratePIR and SearchEquiClass, the algorithm will then visit $\mathcal{C}$'s next unvisited child node which is right after $\mathcal{C}_j$ in the child node list of $\mathcal{C}$.

4.2 Implementation

The pseudo-code of GeneratePIR($\mathcal{C}_j$, $\mathcal{C}$) and SearchEquiClass($\mathcal{C}_j$, $\mathcal{C}$) is shown by Algorithm 1 and 2, respectively.

Algorithm 1 GeneratePIR($\mathcal{C}_j$, $\mathcal{C}$)

```
for each child node $\mathcal{C}_i \neq \mathcal{C}_j$ of $\mathcal{C}$ do
    if $\mathcal{C}_i = (A_i, B_i)$ is unvisited then
        put an unvisited node $(A_j \cap A_i, B_i)$ under $\mathcal{C}_j$;
        put a shadow node $(A_j \cap A_i, |B_j|)$ under $\mathcal{C}_i$; {This is a shadow node for $(A_j \cap A_i, (A_j \cap A_i)')$.}
    else if $\mathcal{C}_i = (A_i, s_i)$ is a shadow node and $A_i$ has not intersected with $A_j$ then
        put a shadow node $(A_j \cap A_i, s_i)$ under $\mathcal{C}_j$;
    end if
end for
mark $\mathcal{C}_j$ as visited;
```

In GeneratePIR, we generate a shadow node $(A_j \cap A_i, s_i)$ under $\mathcal{C}_j$ when $\mathcal{C}_i$ is a shadow node. To obtain a pointer to the concept for $(A_j \cap A_i, s_i)$, we can either build a trie or a hash table for the object sets of concepts generated so far to facilitate the search. Note that if we are only going to generate all concepts, we do not need pointers in shadow nodes or a trie or a hash table for the object sets. To find the equivalence classes in SearchEquiClass yet, for $\mathcal{C}_j$, we build a trie for object sets of its child nodes and put node ids in the leaves of the trie. Then we are able to find the equivalence classes of child nodes by just checking leaves of the trie.

With GeneratePIR and SearchEquiClass ready, we present our lattice construction algorithm in Algorithm 3, where the supremum of the lattice is $\mathcal{U}$.

At the beginning of Algorithm 3, we scan the input matrix once to obtain $\mathcal{U} = (G, G')$ and (A_j, m_j) for each $j \in [1..|M|]$, where m_j is the attribute shared by each member of the object set A_j. For the object sets $A_j \subset G, j \in [1..|M|]$, we build a trie for them and find equivalence classes just as we did in SearchEquiClass. This way we obtain $PIR^{\mathcal{U}}$ as the child nodes of U. Then we construct the remaining of the lattice by processing each node once with GeneratePIR and SearchEquiClass, when we traverse the lattice in DFS with the stack S. In DFS, only unvisited nodes will be pushed into S.

Algorithm 2 SearchEquiClass($\mathcal{C}_j$, $\mathcal{C}$)

```
group $\mathcal{C}_j = (A_j, B_j)$'s child nodes according to their object set to find equivalence
classes $[j_h], h \in [1..r]$;
if $|A_{j_h}| == |A_j|$ for the largest $|A_{j_h}|$ then
  $A_j^{'} = B_{j_h} \cup (int(\mathcal{C}) \cup B_j)$;
else
  $A_j^{'} = int(\mathcal{C}) \cup B_j$;
  mark $\mathcal{C}_j$ as an immediate successor of $C$;
end if
for each equivalence class $[j_h]$ where $|A_{j_h}| \neq |A_j|$ do
  if all child nodes $(A_{j_h}, B_i)$ of $\mathcal{C}_j$ are unvisited then
    remove all $(A_{j_h}, B_i)$ from $\mathcal{C}_j$'s child node list;
    put an unvisited node $(A_{j_h}, B_{j_h})$ under $\mathcal{C}_j$; {This node represents a new con-
    cept.}
  else
    remove all $(A_{j_h}, s_i)$ and $(A_{j_h}, B_i)$ from $\mathcal{C}_j$'s child node list;
    $s = \sum_{(A_{j_h}, s_i)} s_i + \sum_{(A_{j_h}, B_i)} |B_i|$;
    put a shadow node $(A_{j_h}, s)$ under $\mathcal{C}_j$; {Suppose the node is the shadow of
    $\mathcal{E} = (A_{j_h}, (A_{j_h})')$.}
    if $s + |A_j^{'}| = |(A_{j_h})'|$ then
      mark $\mathcal{E}$ as an immediate successor of $\mathcal{C}_j$;
    end if
  end if
end for
sort $\mathcal{C}_j$'s child nodes by their object set size in the ascending order;
```

Algorithm 3 Construct a lattice in the full representation

```
generate $PIR^{\mathcal{U}}$ as $\mathcal{U}$'s child nodes;
sort $\mathcal{U}$'s child nodes by their object set size in the ascending order;
initialize an empty stack $S$;
push(($\mathcal{E}$, $\mathcal{U}$), $S$) where $\mathcal{E}$ is the first node in $\mathcal{U}$'s sorted child node list;
while $S$ is not empty do
  ($\mathcal{C}_j$, $\mathcal{C}$) = pop($S$);
  GeneratePIR($\mathcal{C}_j$, $\mathcal{C}$);
  SearchEquiClass($\mathcal{C}_j$, $\mathcal{C}$);
  if there is an unvisited node $\mathcal{C}_k$ after $\mathcal{C}_j$ among $\mathcal{C}$'s child nodes then
    push(($\mathcal{C}_k$, $\mathcal{C}$), $S$);
  end if
  push(($\mathcal{E}$, $\mathcal{C}_j$), $S$) where $\mathcal{E}$ is the first unvisited node among $\mathcal{C}_j$'s child nodes;
end while
```

4.3 Algorithm analysis

We will show that our algorithm correctly constructs the lattice and the delay-time complexity is $O(|G||M|)$.

Lemma 4.1. *After the completion of the algorithm, For each concept, its extent and intent are correctly computed and its immediate successors are correctly marked in its child nodes.*

The lemma can be proved by applying Theorem 3.1 and Corollary 3.3 in GeneratePIR and SearchEquiClass to check the correctness of $ext(\mathcal{C}_j)$, $int(\mathcal{C}_j)$, and $PIR^{\mathcal{C}_j}$.

The complexity analysis of the algorithm is shown by the following result:

Theorem 4.1. *The algorithm correctly builds the lattice with $O(|G||M|)$ delay time.*

The sketch of the proof can be outlined as follows: The initialization, one run of GeneratePIR and one run of SearchEquiClass all take time $O(|G||M|)$. At the beginning of each iteration of while loop, a new concept is obtained by popping it from the stack. During each iteration of while loop, GeneratePIR and SearchEquiClass are executed once. So it takes $O(|G||M|)$ time to complete one iteration.

As for the space required to store shadow nodes, we can analyze it as follows: Each shadow node needs space $O(|G|)$. The size of the maximal chain in the lattice is at most $|G|$. At each level of the chain, the algorithm will incur at most $O(|M|^2)$ shadow nodes. When the algorithm reaches the bottom of the chain, the size of space is maximized, which is $O(|G|^2|M|^2)$. Other previously used shadow nodes are already recycled in SearchEquiClass. So we only need $O(|G|^2|M|^2)$ space for storing shadow nodes.

5 Algorithm for the Compact Representation

We will define the compact representation for lattices and modify the above algorithm to construct the lattice in a compact representation.

5.1 Compact Representation

Usually a lattice is represented in the full representation as in Figure 2 (a). It is easy to see there is much redundant information in this representation. Suppose (A_1, B_1) is a successor of (A_2, B_2) and $A_1 \subset A_2$. Given (A_2, B_2), we can represent the concept (A_1, B_1) as $(A_2 \setminus A_1, B_1 \setminus B_2)$. Following this idea, we define the compact representation of the lattice $\mathcal{L}$ as follows:

Definition 5.1. *For each concept $\mathcal{C} = (A_1, B_1)$ in $\mathcal{L}$, the* compact representation *of $\mathcal{C}$ regarding $\mathcal{D} = (A_2, B_2)$ is $(A_2 \setminus A_1, B_1 \setminus B_2)$, where $\mathcal{C} \preceq \mathcal{D}$. $\mathcal{D}$ is called $\mathcal{C}$'s* base*.*

We denote such a compact representation of $\mathcal{C}$ as $(CR_{\mathcal{D}}(ext(\mathcal{C})), CR_{\mathcal{D}}(int(\mathcal{C})))$. In Figure 2 (b), we show a concept lattice in a compact representation.

Note that the compact representation is not unique and depends on how we choose the base for each concept. As long as we keep the identity of the base for each concept, we are able to recover the full representation from the compact one, in which we only need to perform two set operations for each concept in a top-down manner.

5.2 Implementation

First, let us consider the compact representation for extents. In [22] a technique using set difference, called diffset, was applied to speed up computation of closed itemsets. The diffset can be used to compute the compact presentation for extents as well.

Suppose nodes $\mathcal{C}_i$ and $\mathcal{C}_j$ are two successors of $\mathcal{C}$. We restate an observation by Zaki *et al.* as follows:

Proposition 5.1. *[22] Suppose $\mathcal{F} = \wedge\{\mathcal{C}_i, \mathcal{C}_j\}$. For $\mathcal{C}_i$ and $\mathcal{C}_j$, if neither of them is an immediate successor of the other, then*
$CR_{\mathcal{C}_j}(ext(\mathcal{F})) = CR_{\mathcal{C}}(ext(\mathcal{C}_i)) \setminus CR_{\mathcal{C}}(ext(\mathcal{C}_j))$.
If $\mathcal{C}_i$ is an immediate successor of $\mathcal{C}_j$, i. e. $\mathcal{F} = \mathcal{C}_i$, then
$CR_{\mathcal{C}_j}(ext(\mathcal{F})) = CR_{\mathcal{C}}(ext(\mathcal{C}_j)) \setminus CR_{\mathcal{C}}(ext(\mathcal{C}_i))$.

To generate a compact representation, we will modify the algorithms for the full representation. In the beginning of Algorithm 3, we will represent the concepts in $IR^{\mathcal{U}}$ in a compact representation with their base to be $\mathcal{U}$. Note that there is no base for $\mathcal{U}$. In GeneratePIR($\mathcal{C}_j$, $\mathcal{C}$), suppose $\mathcal{C}_j = (CR_{\mathcal{C}}(ext(\mathcal{C}_j)), B_j)$ and $\mathcal{C}_i = (CR_{\mathcal{C}}(ext(\mathcal{C}_i)), B_i)$ and they are two child nodes under $\mathcal{C}$. Moreover, suppose $\mathcal{F} = \wedge\{\mathcal{C}_i, \mathcal{C}_j\}$. We only need to do the following: We will put $CR_{\mathcal{C}_j}(ext(\mathcal{F}))$ into the node under $\mathcal{C}_j$, and put $CR_{\mathcal{C}_i}(ext(\mathcal{F}))$ into the node under $\mathcal{C}_i$. The computation of $CR_{\mathcal{C}_j}(ext(\mathcal{F}))$ and $CR_{\mathcal{C}_i}(ext(\mathcal{F}))$ will use $CR_{\mathcal{C}}(ext(\mathcal{C}_j))$ and $CR_{\mathcal{C}}(ext(\mathcal{C}_i))$. Details about the computation will be provided later. In SearchEquiClass($\mathcal{C}_j$, $\mathcal{C}$), at the beginning we compute $int(\mathcal{C}_j)$ and thus generate the concept $\mathcal{C}_j$. To compute $\mathcal{C}_j$'s compact representation, $\mathcal{C}$ is set as its base. $CR_{\mathcal{C}}(ext(\mathcal{C}_j))$ is already obtained in previous run of GeneratePIR. And $CR_{\mathcal{C}}(int(\mathcal{C}_j)) = B_j$. In addition, the operation related to the size of object sets needs to be modified. For example, child nodes should be sorted in descending order by the size of the compact representation of their extents.

Actually, the diffset technique can be improved to make it more efficient to compute extents in a compact representation with the help of shadow nodes. Suppose there are 3 child nodes $\mathcal{C}_i, i \in [1..3]$ under a node. $|ext(\mathcal{C}_3)|$ is the smallest among the three extents. Furthermore, suppose $\mathcal{D} = \wedge\{\mathcal{C}_1, \mathcal{C}_3\}$, $\mathcal{D}^* = \wedge\{\mathcal{C}_2, \mathcal{C}_3\}$ and $\mathcal{E} = \wedge\{\mathcal{C}_1, \mathcal{C}_2\}$. When we visit the node $\mathcal{C}_3$, we will put a shadow node corresponding $\mathcal{D}$ containing $CR_{\mathcal{C}_1}(ext(\mathcal{D}))$ under $\mathcal{C}_1$ and a shadow node corresponding $\mathcal{D}^*$ containing $CR_{\mathcal{C}_2}(ext(\mathcal{D}^*))$ under $\mathcal{C}_2$. We may continue to apply Proposition 5.1 to compute $CR_{\mathcal{C}_j}(ext(\mathcal{E}))$ (j is 1 or 2) by using $CR_{\mathcal{C}}(ext(\mathcal{C}_1))$ and $CR_{\mathcal{C}}(ext(\mathcal{C}_2))$. However, since $|CR_{\mathcal{C}_1}(ext(\mathcal{D}))| \leq |CR_{\mathcal{C}}(ext(\mathcal{C}_1))|$ and $|CR_{\mathcal{C}_1}(ext(\mathcal{D}^*))| \leq |CR_{\mathcal{C}}(ext(\mathcal{C}_2))|$, we are interested in how to compute $CR_{\mathcal{C}_j}(ext(\mathcal{E}))$ (j is 1 or 2) more efficiently with the help of two shadow nodes. As the following lemma shows, when $\mathcal{D} = \mathcal{D}^*$, which often occurs during lattice construction, we can completely avoid using $CR_{\mathcal{C}}(ext(\mathcal{C}_1))$ and $CR_{\mathcal{C}}(ext(\mathcal{C}_2))$.

Lemma 5.1. *Suppose* $\mathcal{D} = \wedge\{\mathcal{C}_1, \mathcal{C}_3\} = \wedge\{\mathcal{C}_2, \mathcal{C}_3\}$*, where* $\mathcal{C}_i, i \in [1..3]$ *are concepts,* $|ext(\mathcal{C}_3)| \leq |ext(\mathcal{C}_1)|$ *and* $|ext(\mathcal{C}_3)| \leq |ext(\mathcal{C}_2)|$*. If* $\mathcal{E} = \wedge\{\mathcal{C}_1, \mathcal{C}_2\}$*, then*
$CR_{\mathcal{C}_1}(ext(\mathcal{E})) = CR_{\mathcal{C}_1}(ext(\mathcal{D})) \setminus CR_{\mathcal{C}_2}(ext(\mathcal{D}))$,
$CR_{\mathcal{C}_2}(ext(\mathcal{E})) = CR_{\mathcal{C}_2}(ext(\mathcal{D})) \setminus CR_{\mathcal{C}_1}(ext(\mathcal{D}))$.

Clearly, the complexity for constructing the compact representation for a lattice is the same as the one for constructing the full representation.

6 Conclusion and future direction

Because of many applications of lattices in various areas, it has become an important question to construct lattices efficiently. Previously, the best delay-time complexity is $O(|G||M|^2)$ for lattice construction algorithms. In this paper, we propose a linear delay algorithm for constructing a lattice with the input matrix of size $|G||M|$. Other advantages of the algorithm include that it does not need the union operation for computing intents of concepts. And it does not check against all generated concepts to see if a new pair of object set and attribute set is a new concept or will be subsumed. In addition, we propose to represent concept lattices in a compact representation, which eliminates redundant information in the full representation. The algorithm for the full representation is modified with improved diffset technique to build a compact representation for lattices.

The lower bound of delay-time complexity for lattice construction algorithms is still unknown. Our future work will focus on finding this lower bound and designing new algorithms to match the bound. With the efficient lattice construction algorithm we also like to apply lattices in more areas.

References

1. Anne Berry, Jean-Paul Bordat, and Alain Aigayret. Concepts can't afford to stammer. In *INRIA Proceedings of the International Conference, Journées de l'Informatique Messine (JIM'03)*, 2003.
2. J.-P. Bordat. Calcul pratique du treillis de galois dune correspondance. *Mathmatique, Informatique et Sciences Humaines*, 24:31–47, 1986.
3. Claudio Carpineto and Giovanni Romano. A lattice conceptual clustering system and its application to browsing retrieval. *Machine Learning*, 24:95–122, 1996.
4. Claudio Carpineto and Giovanni Romano. *Concept Data Analysis: Theory and Applications.* Wiley, 2004.

5. Vicky Choi. Faster algorithms for constructing a concept (galois) lattice. `http://arxiv.org/pdf/cs.DM/0602069`, 2006.
6. Vicky Choi, Yang Huang, Vy Lam, Dustin Potter, Reinhard Laubenbacher, and Karen Duca. Using formal concept analysis for microarray data comparison. In *Proceedings of the 5th Asia Pacific Bioinformatics Conference*, pages 57–66, 2006.
7. David Eppstein. Arboricity and bipartite subgraph listing algorithm. *Information Processing Letters*, 54:207–211, 1994.
8. Timothy J. Everts, Sung Sik Park, and Byeong Ho Kang. Using formal concept analysis with an incremental knowledge acquisition system for web document management. In *Proceedings of the 29th Australasian Computer Science Conference*, pages 247–256, 2006.
9. B. Ganter and R. Wille. *Formal concept analysis: Mathematical Foundations.* Springer, Heidelberg, 1999.
10. Robert Godin, Rokia Missaoui, and Hassan Alaoui. Incremental concept formation algorithms based on galois (concept) lattices. *Computational Intelligence*, 11:246–267, 1995.
11. Yang Huang and Martin Farach-Colton. Lattice based clustering of temporal gene-expression matrices. In *Proceedings of the 7th SIAM International Conference on Data Mining*, 2007.
12. David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, 1988.
13. Y. Kalfoglou, S. Dasmahapatra, and Y. Chen-Burger. Fca in knowledge technologies: Experiences and opportunities. In *Proceedings of 2nd International Conference on Formal Concept Analysis*, pages 252–260. Springer, 2004.
14. K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Proceedings of 9th Scand. Workshop on Algorithm Theory*, pages 260–272, 2004.
15. L. Nourine and O. Raynaud. A fast algorithm for building lattices. *Information Processing Letters*, 71:199–204, 1999.
16. Gregor Snelting and Frank Tip. Reengineering class hierarchies using concept analysis. In *Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 99–110, 1998.
17. Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *IEEE ICDM Workshop on Frequent Itemset Mining Implementation*, 2004.
18. J. Wang, J. Han, and J. Pei. Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 344–353, 2004.
19. Guizhen Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 344–353, 2004.
20. M. J. Zaki and C.-J. Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transaction on Knowledge and Data Engineering*, 17:462–478, 2005.
21. M. J. Zaki and M. Ogihara. Theoretical foundations of association rules. In *Proceedings of 3rd ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 1–7, 1998.
22. Mohammed J. Zaki and Karam Gouda. Fast vertical mining using diffsets. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 326–335, 2003.