Merging Closed Pattern Sets in Distributed Multi-Relational Data

Hirohisa Seki* and Yohei Kamiya

Dept. of Computer Science, Nagoya Inst. of Technology,
Showa-ku, Nagoya 466-8555, Japan
seki@nitech.ac.jp

Abstract. We consider the problem of mining closed patterns from multi-relational databases in a distributed environment. Given two local databases (*horizontal* partitions) and their sets of closed patterns (concepts), we generate the set of closed patterns in the global database by utilizing the *merge* (or *subposition*) operator, studied in the field of Formal Concept Analysis. Since the execution times of the merge operations increase with the increase in the number of local databases, we propose some methods for improving the merge operations. We also present some experimental results using a distributed computation environment based on the MapReduce framework, which shows the effectiveness of the proposed methods.

Key Words: multi-relational data mining, closed patterns, merge (subposition) operator, FCA, distributed databases, MapReduce

1 Introduction

Multi-relational data mining (MRDM) has been extensively studied for more than a decade (e.g., [7, 8] and references therein), and is still attracting increasing interest in the fields of data mining (e.g., [14, 29]) and inductive logic programming (ILP). In the framework of MRDM, data and patterns (or queries) are represented in the form of logical formulae such as datalog (a class of first order logic). This expressive formalism of MRDM allows us to use complex and structured data in a uniform way, including trees and graphs in particular, and multi-relational patterns in general.

On the other hand, Formal Concept Analysis (FCA) has been developed as a field of applied mathematics based on a clear mathematization of the notions of concept and conceptual hierarchy [11]. While it has attracted much interest from various application areas including, among others, data mining, knowledge acquisition and software engineering (e.g., [12]), research on extending the capabilities of FCA for AI (Artificial Intelligence) has recently been attracted much attention [20].

* This work was partially supported by JSPS Grant-in-Aid for Scientific Research (C) 24500171.

The notion of *iceberg query lattices*, proposed by Stumme [30], combines the notions of MRDM and FCA; frequent datalog queries in MRDM correspond to iceberg concept lattices (or *frequent closed itemsets*) in FCA. Ganter and Kuznetsov [10] have extensively studied the framework of more expressive pattern structures. In MRDM, condensed representations such as closed patterns and free patterns have been also studied in *c-armr* by De Raedt and Ramon [6], and in RelLCM2 by Garriga et al. [13].

We consider in this paper the problem of mining closed patterns (or queries) in multi-relational data, particularly applying the notion of iceberg query lattices to a *distributed* mining setting. The assumption that a given dataset is distributed and stored in different sites will be reasonable for some situations where we might not be able to move local datasets into a centralized site due to too much data size and/or privacy concerns.

Given two local databases (*horizontal* partitions) and their sets of closed patterns (concepts), the set of closed patterns in the global database can be constructed by using *subposition*) operator [11, 33] or the *merge* operator [23]. From our preliminary experiments [28] using a distributed computation environment MapReduce [3], we have found that the execution times of computing the merge operations have increased with the increase in the number of local databases. In this paper, we therefore propose some methods for computing the merge operations so that we can efficiently construct the set of global closed patterns from the sets of local closed patterns. Our methods are based on the properties of the merge operator.

The organization of the rest of this paper is as follows. After summarizing some basic notations and definitions of closed patterns mining in MRDM in Sect. 2, we consider distributed closed pattern mining in MRDB and the merge operator in Sect. 3. We then explain our approach to improving the merge operations in Sect. 4. In Section 5, we show the effectiveness of our methods by some experimental results. Finally, we give a summary of this work in Section 6.

2 Iceberg Query Lattices in Multi-Relational Data Mining

2.1 Multi-Relational Data Mining

In the task of frequent pattern mining in multi-relational databases, we assume that we have a given database **r**, a language of patterns, and a notion of frequency which measures how often a pattern occurs in the database. We use datalog, or Prolog without function symbols other than constants, to represent data and patterns. We assume some familiarity with the notions of logic programming (e.g., [22, 24]), although we introduce some notions and terminology in the following.

Example 1. Consider a multi-relational database **r** in Fig. 1 (above), which consists of five relations, Customer, Parent, Buys, Male and Female. For each relation, we introduce a corresponding predicate, i.e., *customer*, *parent*, *buys*, *male* and *female*, respectively.

Customer

key
allen
carol
diana
fred

Parent

SR.	*JR.*
allen	bill
allen	jim
carol	bill
diana	eve
fred	eve
fred	hera

Buys

key	*item*
allen	pizza
carol	pizza
diana	cake
fred	cake

Male

person
bill
jim

Female

person
eve
hera

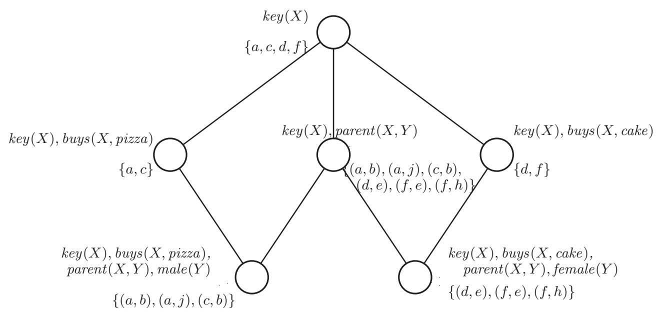


Fig. 1. An Example of Datalog Database **r** with customer relation as a key (above) and the Iceberg Query Lattice Associated to **r** (below), where a substitution $\theta = \{X/t_1, Y/t_2\}$ (resp., $\theta = \{X/t_1\}$) is simply denoted by (t_1, t_2) (resp., t_1), and the name (e.g., *allen*) of each person in the tables is abbreviated to its first character (e.g., a).

Consider the following pattern $P = customer(X), parent(X,Y), buys(X, pizza)$. For a substitution θ, $P\theta$ is logically entailed by $\mathbf{r}$, denoted by $\mathbf{r} \models P\theta$, if there exists a tuple (a_1, a_2) such that $a_1 \in$ Customer, $(a_1, a_2) \in$ Parent, and tuple $(a_1, pizza) \in$ Buys. Then, $answerset(P, \mathbf{r}) = \{\{X/allen, Y/bill\}, \{X/allen, Y/jim\}, \{X/carol, Y/bill\}\}$. □

An *atom* (or *literal*) is an expression of the form $p(t_1, \ldots .t_n)$, where p is a *predicate* (or *relation*) of arity n, denoted by p/n, and each t_i is a *term*, i.e., a constant or a variable.

A substitution $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ is an assignment of terms to variables. The result of applying a substitution θ to an expression E is the expression $E\theta$, where all occurrences of variables V_i have been simultaneously replaced by the corresponding terms t_i in θ. The set of variables occurring in E is denoted by $Var(E)$.

A *pattern* is expressed as a conjunction of atoms (literals) $l_1 \wedge \cdots \wedge l_n$, denoted simply by $l_1, \ldots, l_n$. A pattern is sometimes called a *query*. We will represent conjunctions in list notation, i.e., $[l_1, \ldots, l_n]$. For a conjunction C and an atom p, we denote by $[C, p]$ the conjunction that results from adding p after the last element of C.

Let C be a pattern (i.e., a conjunction) and θ a substitution of $\mathit{Var}(C)$. When $C\theta$ is logically entailed by a database $\mathbf{r}$, we write it by $\mathbf{r} \models C\theta$. Let $\mathit{answerset}(C, \mathbf{r})$ be the set of substitutions satisfying $\mathbf{r} \models C\theta$.

In multi-relational data mining, one of the predicates is often specified as a *key* (or *target*) (e.g., [4, 6]), which determines the entities of interest and what is to be counted. The key (target) is thus to be present in all patterns considered. In Example 1, the key is predicate *customer*.

Let $\mathbf{r}$ be a database and Q be a query containing a key atom $\mathit{key}(X)$. Then, the *support* (or *frequency*) of Q, denoted by $\mathit{supp}(Q, \mathbf{r}, \mathit{key})$, is defined to be the number of different keys that answer Q (called the *support count* or *absolute support*), divided by the total number of keys. Q is said to be *frequent*, if $\mathit{supp}(Q, \mathbf{r}, \mathit{key})$ is no less than some user defined threshold $\mathit{min\_sup}$.

A pattern containing a key will not be always meaningful; for example, let $C = [\mathit{customer}(X), \mathit{parent}(X, Y), \mathit{buys}(Z, \mathit{pizza})]$ be a conjunction in Example 1. Variable Z in C is not *linked* to variable X in key atom $\mathit{customer}(X)$; an object represented by Z will have nothing to do with key object X. It will be inappropriate to consider such a conjunction as an intended pattern to be mined. In ILP, the following notion of *linked literals* [16] is used to specify the so-called *language bias*.

Definition 1 (Linked Literal). [16] Let $\mathit{key}(X)$ be a key atom and l a literal. l is said to be *linked* to $\mathit{key}(X)$, if either $X \in \mathit{Var}(l)$ or there exists a literal l_1 such that l is linked to $\mathit{key}(X)$ and $\mathit{Var}(l_1) \cap \mathit{Var}(l) \neq \emptyset$. $\square$

Given a database $\mathbf{r}$ and a key atom $\mathit{key}(X)$, we assume that there are predefined finite sets of predicate (resp. variables; resp. constant symbols), and that, for each literal l in a conjunction C, it is constructed using the predefined sets. Moreover, each pattern C of conjunctions satisfies the following conditions: $\mathit{key}(X) \in C$ and, for each $l \in C, l$ is linked to $\mathit{key}(X)$. In the following, we denote by $\mathcal{Q}$ the set of queries (or patterns) satisfying the above bias condition.

2.2 Iceberg Query Lattices with Key

We now consider the notion of a formal context in MRDM, following [30].

Definition 2. [30] Let $\mathbf{r}$ be a datalog database and $\mathcal{Q}$ a set of datalog queries. The *formal context associated to* $\mathbf{r}$ and $\mathcal{Q}$ is defined by $\mathcal{K}_{\mathbf{r},\mathcal{Q}} = (O_{\mathbf{r},\mathcal{Q}}, A_{\mathbf{r},\mathcal{Q}}, I_{\mathbf{r},\mathcal{Q}})$, where $O_{\mathbf{r},\mathcal{Q}} = \{\theta \mid \theta$ is a grounding substitution for all $Q \in \mathcal{Q}\}$, and $A_{\mathbf{r},\mathcal{Q}} = \mathcal{Q}$, and $(\theta, Q) \in I_{\mathbf{r},\mathcal{Q}}$ if and only if $\theta \in \mathit{answerset}(Q, \mathbf{r})$. $\square$

From this formal context, we can define the concept lattice the same way as in [30]. We first introduce an equivalence relation $\sim_{\mathbf{r}}$ on the set of queries: Two queries Q_1 and Q_2 are said to be *equivalent* with respect to database $\mathbf{r}$ if and only if $\mathit{answerset}(Q_1, \mathbf{r}) = \mathit{answerset}(Q_2, \mathbf{r})$. We note that $\mathit{Var}(Q_1) = \mathit{Var}(Q_2)$ when $Q_1 \sim_{\mathbf{r}} Q_2$.

Definition 3 (Closed Query). Let $\mathbf{r}$ be a datalog database and $\sim_{\mathbf{r}}$ the equivalence relation on a set of datalog queries $\mathcal{Q}$. A query (or pattern) Q is said to be *closed* (w.r.t. $\mathbf{r}$ and $\mathcal{Q}$), iff Q is the most specific query among the equivalence class to which it belongs: $\{Q_1 \in \mathcal{Q} \mid Q \sim_{\mathbf{r}} Q_1\}$. □

For any query Q_1, its *closure* is a closed query Q such that Q is the most specific query among $\{Q \in \mathcal{Q} \mid Q \sim_{\mathbf{r}} Q_1\}$. Since it uniquely exists, we denote it by $Clo(Q_1; \mathbf{r})$. We note again that $Var(Q_1) = Var(Clo(Q_1; \mathbf{r}))$ by definition. We refer to this as the *range-restricted* condition here.

Stumme [30] showed that the set of frequent closed queries forms a lattice, called *an iceberg query lattice*. In our framework, it is necessary to take our bias condition into consideration. To do that, we employ the well-known notion of the most specific generalization (or *least generalization*) [26, 24].

For queries Q_1 and Q_2, we denote by $lg(Q_1, Q_2)$ the least generalization of Q_1 and Q_2. Moreover, the *join* of Q_1 and Q_2, denoted by $Q_1 \vee Q_2$, is defined as: $Q_1 \vee Q_2 = lg(Q_1, Q_2)|_{\mathcal{Q}}$, where, for a query Q, $Q|_{\mathcal{Q}}$ is the *restriction* of Q to $\mathcal{Q}$, defined by a conjunction consisting of every literal l in Q which is linked to $key(X)$, i.e., deleting every literal in Q not linked to $key(X)$.

Definition 4. [30] Let $\mathbf{r}$ be a datalog database and $\mathcal{Q}$ a set of datalog queries. The *iceberg query lattice associated to* $\mathbf{r}$ and $\mathcal{Q}$ for $minsupp \in [0, 1]$ is defined as: $\mathcal{C}_{\mathbf{r},\mathcal{Q}} = (\{Q \in \mathcal{Q} \mid Q \text{ is closed w.r.t. } \mathbf{r} \text{ and } \mathcal{Q}, \text{ and } Q \text{ is frequent}\}, \models)$, where $\models$ is the usual logical implication. □

Example 2. Fig. 1 (below) shows the iceberg query lattice associated to $\mathbf{r}$ in Ex. 1 and $\mathcal{Q}$ with the support count 1, where each query $Q \in \mathcal{Q}$ has $customer(X)$ as a key atom, denoted by $key(X)$ for short, Q is supposed to contain at most two variables (i.e., X, Y), and the 2nd argument of predicate *buys* is a constant. □

Theorem 1. [28] Let $\mathbf{r}$ be a datalog database and $\mathcal{Q}$ a set of datalog queries where all queries contain an atom *key* and they are linked. Then, $\mathcal{C}_{\mathbf{r},\mathcal{Q}}$ is a $\vee$-semi-lattice. □

3 Distributed Closed Pattern Mining in MRDB

Horizontal Decomposition of MRDB and Mining Local Concepts

Our purpose in this work is to mine global concepts in a distributed setting, where a global database is supposed to be horizontally partitioned appropriately, and stored possibly in different sites. We first consider the notion of a *horizontal decomposition* of a multi-relational DB. Since a multi-relational DB consists of multiple relations, its horizontal decomposition is not immediately clear.

Definition 5. Let $\mathbf{r}$ be a multi-relational datalog database with a key predicate *key*. We call a pair $\mathbf{r}_1, \mathbf{r}_2$ a *horizontal decomposition* of $\mathbf{r}$, if (i) $\mathrm{key}_{\mathbf{r}} = \mathrm{key}_{\mathbf{r}_1} \uplus \mathrm{key}_{\mathbf{r}_2}$, i.e., the key relation $\mathrm{key}_{\mathbf{r}}$ in $\mathbf{r}$ is disjointly decomposed into $\mathrm{key}_{\mathbf{r}_1}$ and $\mathrm{key}_{\mathbf{r}_2}$ in $\mathbf{r}_1$ and $\mathbf{r}_2$, respectively, and (ii) for any query Q, $answerset(Q, \mathbf{r}) = answerset(Q, \mathbf{r}_1) \cup answerset(Q, \mathbf{r}_2)$. □

The second condition in the above states that the relations other than the key relation in $\mathbf{r}$ are decomposed so that any answer substitution in $answerset(Q, \mathbf{r})$ is computed either in partition $\mathbf{r}_1$ or $\mathbf{r}_2$, thereby being preserved in this horizontal decomposition. An example of a horizontal decomposition of $\mathbf{r}$ is shown in Example 3 below.

Given a horizontal decomposition of a multi-relational DB, we can utilize any preferable concept (or closed pattern) mining algorithm for computing local concepts on each partition, as long as the mining algorithm is applicable to MRDM and its resulting patterns satisfy our bias condition. We use here an algorithm called *ffCLM* [27], which is based on the notion of *closure extension* due to Pasquier et al. [25] and Uno et al. [32] in frequent itemset mining.

Computing Global Closed Patterns by Merge Operator in MRDM

To compute the set of global closed patterns from the sets of local closed patterns in MRDM, we need the following *merge* operator $\oplus$. For patterns C_1 and C_2, we denote by $C_1 \cap C_2$ a possibly empty conjunction of the form: $l_1 \wedge \cdots \wedge l_k$ $(k \geq 0)$ such that, for each l_i $(i \leq k)$, $l_i \in C_1$ and $l_i \in C_2$.

Theorem 2. *[28] Let* $\mathbf{r}$ *be a datalog database, and* $\mathbf{r}_1, \mathbf{r}_2$ *a horizontal decomposition of* $\mathbf{r}$. *Let* $\mathcal{C}$ $(\mathcal{C}_i)$ $(i = 1, 2)$ *be the set of closed patterns of* $\mathbf{r}$ $(\mathbf{r}_i)$*, respectively. Then, we have the following:*

$$\begin{aligned}\mathcal{C} &= \mathcal{C}_1 \oplus \mathcal{C}_2 \\ &= (\mathcal{C}_1 \cup \mathcal{C}_2) \cup \{C_1 \cap C_2 \mid C_1 \in \mathcal{C}_1, C_2 \in \mathcal{C}_2, \\ &\qquad C_1 \cap C_2 \textit{ is linked with key.}\} \end{aligned} \quad (1)$$

The set of global closed patterns $\mathcal{C}$ is obtained by the union of the local closed patterns $\mathcal{C}_1$ and $\mathcal{C}_2$, and, in addition to that, by intersecting each pattern $C_1 \in \mathcal{C}_1$ and $C_2 \in \mathcal{C}_2$. Furthermore, the pattern obtained by the intersection, $C_1 \cap C_2$, should satisfy the bias condition (Def. 1). We note that $C_1 \cap C_2$ does not necessarily satisfy the linkedness condition; for example, suppose that C_1 (C_2) is a closed pattern of the form: $C_1 = key(X), p(X, Y), m(Y)$ $(C_2 = key(X), q(X, Y), m(Y))$, respectively. Then, $C_1 \cap C_2 = key(X), m(Y)$, which is not linked to $key(X)$, and thus does not satisfy the bias condition.

We note that, in the case of transaction databases, the above theorem coincides with the one by Lucchese et al. [23].

Example 3. We consider a horizontal decomposition $\mathbf{r}_1, \mathbf{r}_2$ of $\mathbf{r}$ in Example 1 such that the key relation $\mathrm{key}_{\mathbf{r}}$ (i.e., Customer) in $\mathbf{r}$ is decomposed into $\mathrm{key}_{\mathbf{r}_1} = \{\mathrm{allen}, \mathrm{carol}\}$ and $\mathrm{key}_{\mathbf{r}_2} = \{\mathrm{dian}, \mathrm{fred}\}$, and the other relations than Customer are decomposed so that they satisfy the second condition of Def. 5.

Consider a globally closed pattern $C = [key(X), parent(X, Y)]$ in Fig. 1. In $\mathbf{r}_1$, there exists a closed pattern C_1 of the form: $[C, buys(X, pizza), male(Y)]$, while, in $\mathbf{r}_2$, there exists a closed pattern C_2 of the form: $[C, buys(X, cake), female(Y)]$. Then, we have that C coincides with $C_1 \cap C_2$. $\square$

We can now formulate our problem as follows:

Mining Globally Closed Patterns from Local DBs:
Input: A set of local databases $\{DB_1, \ldots, DB_n\}$
Output: the set of global closed patterns $\mathcal{C}_{1..n}$.

In order to compute $\mathcal{C}_{1..n}$, our approach consists of two phases: we first compute each set $\mathcal{C}_i$ $(i = 1, \ldots, n)$ of local closed patterns from DB_i, and then we compute $\mathcal{C}_{1..n}$ by applying the merge operators. We call the first phase the *mining phase*, while we call the second phase the *merge phase*.

4 Making Merge Computations Efficient in MRDM

In the merge operation in conventional data mining such as itemsets, computing the intersection of two sets in the merge operation $\oplus$ is straightforward. In MRDM, on the other hand, the computation of $\oplus$ operator becomes somewhat involved due to handling variables occurring in patterns. Namely, two additional tests are required: checking the bias condition (linkedness), and checking equivalence modulo variable renaming for eliminating duplicate patterns.

For closed patterns C_1 and C_2, we must check whether the intersection $C_1 \cap C_2$ satisfies the linkedness condition. Moreover, we must check whether $C_1 \cap C_2$ is equivalent (modulo variable renaming) to the other patterns obtained so far. For example, let C_1 (C_2) be a pattern of the form: $C_1 = \mathit{key}(X), p(X,Y), m(Y)$ $(C_2 = \mathit{key}(X), p(X,Z), m(Z))$, respectively. Then, C_1 is equivalent to C_2 modulo variable renaming.

When implementing a data mining system, such handling variables in patterns will necessarily require string manipulations, and such string operations would lead to undesirable overhead in actual implementation. In the following, we therefore propose two methods for reducing the computational costs in the merge operation.

4.1 Partitioning Pattern Sets

When computing the merge operation, we can use the following property:

Proposition 1. *Let $DB = DB_1 \cup DB_2$, and $\mathcal{C}$ $(\mathcal{C}_i)$ the set of closed patterns of DB (DB_i) $(i = 1, 2)$, respectively. Then,*

$$\begin{aligned}\mathcal{C} &= \mathcal{C}_1 \oplus \mathcal{C}_2 \\ &= (\mathcal{C}_1 \cup \mathcal{C}_2) \cup \{C_1 \cap C_2 \mid (C_1, C_2) \in (\mathcal{C}_1, \mathcal{C}_2)\,, \\ &\quad\; C_1 \cap C_2 : \textit{linked with key}, \mathit{Var}(C_1) = \mathit{Var}(C_2)\}\end{aligned} \tag{2}$$

Proof. Let C be a closed pattern in $\mathcal{C}$ such that C is linked with key. From Theorem 2, it suffices to show that there exist patterns $C_i \in \mathcal{C}_i$ $(i = 1, 2)$ such that $C = C_1 \cap C_2$ and $\mathit{Var}(C_1) = \mathit{Var}(C_2)$.

Let $C_i = \mathit{Clo}(C; DB_i)$ $(i = 1, 2)$. Then, we have from the definition of $\mathit{Clo}(\cdot;\cdot)$ that $\mathit{Var}(C) = \mathit{Var}(C_1) = \mathit{Var}(C_2)$. Moreover, we can show that $C = C_1 \cap C_2$, which is to be proved. □

From the above proposition, when computing the intersection of each pair of patterns $C_1 \in \mathcal{C}_1$ and $C_2 \in \mathcal{C}_2$ in (1), we can perform the intersection of only those pairs (C_1, C_2) containing the same set of variables, i.e., $\mathit{Var}(C_1) = \mathit{Var}(C_2)$. When compared with the original definition of the merge operator $\oplus$ (Theorem 2), the above property will be utilized to reduce the cost of the merge operations.

4.2 Merging Diff-Sets

Next, we consider another method for making the merge operation efficient, which is based on the following simple observation:

Observation 1. *Given sets of closed patterns $\mathcal{C}_1$ and $\mathcal{C}_2$, let $\mathcal{D}_1 = \mathcal{C}_1 \setminus \mathcal{C}_2$ and $\mathcal{D}_2 = \mathcal{C}_2 \setminus \mathcal{C}_1$, namely, $\mathcal{D}_i$ is a difference set (*diff-set *for short) $(i = 1, 2)$. Suppose that C is a* new *(or* generator *[33]) pattern in $\mathcal{C}_1 \oplus \mathcal{C}_2$, meaning that $C \in \mathcal{C}_1 \oplus \mathcal{C}_2$, while $C \notin \mathcal{C}_1 \cup \mathcal{C}_2$. Then, C is obtained by intersection operation, i.e., $C = C_1 \cap C_2$ for some patterns $C_1 \in \mathcal{D}_1$ and $C_2 \in \mathcal{D}_2$.*

That is, a new closed pattern C will be generated only when intersecting those patterns in the difference sets in $\mathcal{D}_1$ and $\mathcal{D}_2$. This fact easily follows from the property that the set of closed patterns is a semi-lattice: suppose otherwise that $C_1 \in \mathcal{D}_1$, while $C_2 \notin \mathcal{D}_2$. Then, $C_2 \in \mathcal{C}_1$. Since both C_1 and C_2 are in $\mathcal{C}_1$, we have that $C = C_1 \cap C_2$ is a closed pattern also in $\mathcal{C}_1$, which implies that C is not a new pattern. Algorithm 1 shows the above-mentioned method based on the difference sets. In the algorithm, the computation of supports (or occurrences) is omitted, which is done similarly in [33].

Algorithm 1: Diff-Set_Merge($\mathcal{C}_1, \mathcal{C}_2$)

```
input : sets of closed patterns $\mathcal{C}_1, \mathcal{C}_2$
output: $\mathcal{C}_{1..2} = \mathcal{C}_1 \oplus \mathcal{C}_2$
1 $\mathcal{C} = \mathcal{C}_1 \cap \mathcal{C}_2$; $\mathcal{D}_1 = \mathcal{C}_1 \setminus \mathcal{C}_2$; $\mathcal{D}_2 = \mathcal{C}_2 \setminus \mathcal{C}_1$;
2 foreach pair $(C_1, C_2) \in \mathcal{D}_1 \times \mathcal{D}_2$ do
3     $C \leftarrow C_1 \cap C_2$;
4     if $C$ satisfies the bias condition and $C \notin \mathcal{C}$ then
5         $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$;
6     end
7 end
8 return $\mathcal{C}$
```

5 Experimental Results

Implementation and Test Data

To see the effectiveness of our approach to distributed mining, we have made some experiments. As for the mining phase, we implemented our approach by using Java 1.6.0_22. Experiments of the phase were performed on 8 PCs with Intel Core i5 processors running at 2.8GHz, 8GB of main memory, and 8MB of L2 cache, working under Ubuntu 11.04. We used Hadoop 0.20.2 using 8 PCs, and 2 mappers working on each PC. On the other hand, experiments of the merging phase were performed on one of the PCs.

We use two datasets, often used in the field of ILP; one is the mutagenesis dataset[1], and the other is an English corpus of the Penn Treebank Project[2].

The mutagenesis dataset, for example, contains 30 chemical compounds. Each compound is represented by a set of facts using predicates such as *atom*, *bond*, for example. The size of the set of predicate symbols is 12. The size of key atom ($active(X)$) is 230, and minimum support $min\_sup = 1/230$. We assume that patterns contain at most 4 variables and they contain no constant symbols. The number of the closed patterns mined is $5,784$.

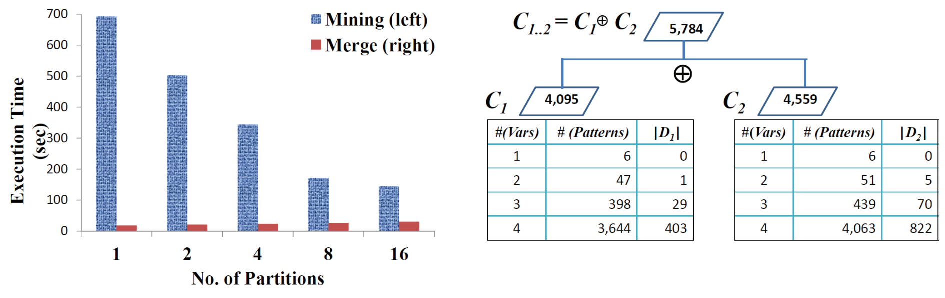
Effect of Partitioning Pattern Sets

Fig. 2 (left) summarizes the results of the execution times for a test data on the mutagenesis dataset. We can see from the figure that the execution times t_1 of the mining phase are reduced almost linearly with the number of partitions. On the other hand, the execution times t_2 of the merging phase for obtaining global closed patterns increase almost linearly with the number p of partitions from 1 (i.e., no partitioning) to 16. This is reasonable; the number of applying the merge operators is $(p-1)$ when we have p partitions. Note that the execution time for the merge phase in the case of a single partition means some start-up overheads such as opening/reading a file of the results of the mining phase, followed by preparing the inputs of the merge operation.

In this particular example, the time spent in the merge phase is relatively small when compared with that for the mining phase. This is because the number of partitions and the number of local closed patterns are rather small. When the number of partitions of a global database becomes larger, however, the execution times for the merging phase will become inevitably larger. Considering efficient merge algorithms is thus an important issue for scalability in MRDM.

To see the effect of using Proposition 1, Fig. 2 (right) shows the numbers of closed patterns in a merge computation $\mathcal{C}_1 \oplus \mathcal{C}_2$ with input sets $\mathcal{C}_1, \mathcal{C}_2$ of closed patterns for the mutagenesis dataset with 16 partitions. Each table shows the number of patterns in $\mathcal{C}_i$ $(i = 1, 2)$ containing k variables for $1 \leq k \leq 4$. The number of computing intersection operations based on Proposition 1 has been reduced to about 80% of that of the original computation. The execution times in Fig. 2 (left) are the results obtained by using this method.

[1] http://www.cs.ox.ac.uk/activities/machlearn/mutagenesis.html
[2] http://www.cis.upenn.edu/ treebank/



#(Vars)	# (Patterns)	$\|D_1\|$
1	6	0
2	47	1
3	398	29
4	3,644	403

#(Vars)	# (Patterns)	$\|D_2\|$
1	6	0
2	51	5
3	439	70
4	4,063	822

Fig. 2. Execution Times of the Mining Phase and the Merge Phase (left) and No. of Patterns in a Merge Computation (right): An Example in the Mutagenesis Dataset. Each number in a quadrangle is the size of a closed pattern set. $\mathcal{D}_1 = \mathcal{C}_1 \setminus \mathcal{C}_2$ and $\mathcal{D}_2 = \mathcal{C}_2 \setminus \mathcal{C}_1$.

Effect of Merging Diff-Sets

Fig. 3 shows its performance results (the execution times), compared with the naive method, using the same datasets, the mutagenesis (left) and the English corpus (right).

In both datasets, the execution times decrease as the number n of the local DBs increases; in particular, when $n = 16$ in the mutagenesis data set, the execution time is reduced to about 43% of that of the naive method. To see the reason of this results, Fig. 2 (right) shows the sizes of the difference sets $\mathcal{D}_1$ and $\mathcal{D}_2$ used in the merge computation $\mathcal{C}_1 \oplus \mathcal{C}_2$ with input sets $\mathcal{C}_1, \mathcal{C}_2$ of the closed patterns.

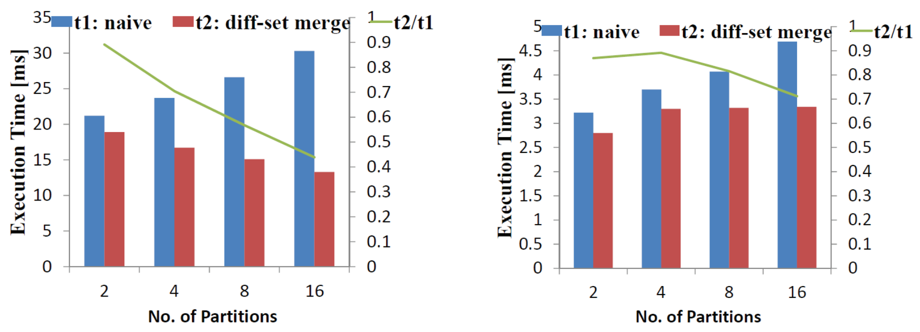


Fig. 3. Results of the Diff-Sets Merge Method: The Mutagenesis Dataset (left) and The English Corpus (right)

6 Concluding Remarks

We have considered the problem of mining closed patterns from multi-relational databases in a distributed environment. For that purpose, we have proposed two methods for making the merge (or subposition) operations efficient, and we have then exemplified the effectiveness of our method by some preliminary experimental results using MapReduce/Hadoop distributed computation framework in the mining process.

In MRDM, efficiency and scalability have been major concerns [2]. Krajca et al. [17, 18] have proposed algorithms to compute search trees for closed patterns simultaneously either in parallel or in a distributed manner. Their approaches are orthogonal to ours; it would be beneficial to employ their algorithms for computing local closed patterns in the mining phase in our framework.

In this work, we have confined ourselves to horizontal partitions of a global MRDB. It will be interesting to study *vertical* partitioning and their mixture in MRDM, where the *apposition* operator studied by Valtchev et al. [34] will play an important role. As future work, our plan is to develop an efficient algorithm dealing with such a general case in MRDM.

Acknowledgement The authors would like to thank anonymous reviewers for their useful comments on the previous version of the paper.

References

1. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. in Proc. VLDB Conf., pp. 487–499, 1994.
2. Blockeel, H., Sebag, M.: Scalability and efficiency in multi-relational data mining. SIGKDD Explorations Newsletter 2003, Vol.4, Issue 2, pp.1-14, 2003.
3. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM, Vol. 51, No. 1, pp.107–113, 2008.
4. Dehaspe, L.: Frequent pattern discovery in first-order logic, PhD thesis, Dept. Computer Science, Katholieke Universiteit Leuven, 1998.
5. Dehaspe, L., Toivonen, H.: Discovery of Relational Association Rules. in S. Dzeroski and N Lavrac (eds.) Relational Data Mining, pp. 189–212, Springer, 2001.
6. De Raedt, L., Ramon, J.: Condensed representations for Inductive Logic Programming. in Proc. KR'04, pp. 438-446, 2004.
7. Dzeroski, S.: Multi-Relational Data Mining: An Introduction. SIGKDD Explorations Newsletter 2003, Vol.5, Issue 1, pp.1-16, 2003.
8. Dzeroski, S., Lavrač, N. (eds.): Relational Data Mining. Springer-Verlag, Inc. 2001.
9. Ganter, B.: Two Basic Algorithms in Concept Analysis, Technical Report FB4-Preprint No. 831, TH Darmstadt, 1984. also in Formal Concept Analysis, LNCS 5986, pp. 312-340, Springer, 2010.
10. Ganter, B., Kuznetsov, S.: Pattern structures and Their Projections, ICCS-01, LNCS, 2120, pp. 129-142, 2001.
11. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer, 1999.

12. Ganter, B., Stumme, G., Wille, R.: Formal Concept Analysis, Foundations and Applications. LNCS 3626, Springer, 2005.
13. Garriga,G. C., Khardon, R., De Raedt, L.: On Mining Closed Sets in Multi-Relational Data. in Proc. IJCAI 2007, pp.804-809, 2007.
14. Goethals, B., Page, W. L., Mampaey, M.: Mining Interesting Sets and Rules in Relational Databases. in Proc. 2010 ACM Sympo. on Applied Computing (SAC '10), pp. 997-1001, 2010.
15. Han, J., Kamber, M.: Data Mining: Concepts and Techniques, 2nd edition, Morgan Kaufmann Publishers Inc., 2005.
16. Helft, N.: Induction as nonmonotonic inference. in Proc. KR'89, pp. 149–156, 1989.
17. Krajca, P., Vychodil, V.: Distributed Algorithm for Computing Formal Concepts Using Map-Reduce Framework, in Proc. IDA '09, Springer-Verlag, pp. 333–344, 2009.
18. Krajca, P., Outrata, J., Vychodil, V.: Parallel algorithm for computing fixpoints of Galois connections, Annals of Mathematics and Artificial Intelligence, Vol. 59, No. 2, pp. 257–272, Kluwer Academic Publishers, 2010.
19. Kuznetsov, S. O.: A Fast Algorithm for Computing All Intersections of Objects in a Finite Semi-lattice, Automatic Documentation and Mathematical Linguistics, Vol. 27, No. 5, pp. 11-21, 1993.
20. Kuznetsov, S. O., Napoli, A., Rudolph, S., eds: FCA4AI: "What can FCA do for Artificial Intelligence?" IJCAI 2013 Workshop, Beijing, China, 2013.
21. Kuznetsov, S. O., Obiedkov, S. A.: Comparing performance of algorithms for generating concept lattices. J. Exp. Theor. Artif. Intell., 14(2-3):189-216, 2002.
22. Lloyd, J. W.: Foundations of Logic Programming, Springer, Second edition, 1987.
23. Lucchese, C., Orlando, S., Rergo, R.: Distributed Mining of Frequent Closed Itemsets: Some Preliminary Results. International Workshop on High Performance and Distributed Mining, 2005.
24. Nienhuys-Cheng, S-H., de Wolf, R.: Foundations of Inductive Logic Programming, LNAI 1228, Springer, 1997.
25. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules. in Proc. ICDT'99, LNAI 3245, pp. 398-416, 1999.
26. Plotkin, G.D.: A Note on Inductive Generalization. Machine Intelligence, Vol. 5, pp. 153-163, 1970.
27. Seki, H., Honda, Y., Nagano, S.: On Enumerating Frequent Closed Patterns with Key in Muti-relational Data. LNAI 6332, pp. 72-86, 2010.
28. Seki, H., Tanimoto, S.: Distributed Closed Pattern Mining in Multi-Relational Data based on Iceberg Query Lattices: Some Preliminary Results. in Proc. CLA'12, pp.115-126, 2012
29. Spyropoulou, E., De Bie. T., Boley, M.: Interesting Pattern Mining in Multi-Relational Data. Data Min. Knowl. Discov. 42(2), pp. 808-849, 2014.
30. Stumme, G.: Iceberg Query Lattices for Datalog. In Conceptual Structures at Work, LNCS 3127, Springer-Verlag, pp. 109-125, 2004.
31. Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing Iceberg Concept Lattices with Titanic. J. on Knowledge and Data Engineering (KDE) 42(2), pp. 189-222, 2002.
32. Uno, T., Asai, T. Uchida, Y., Arimura, H.: An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases. DS'04, LNAI 3245, pp. 16-31, 2004.
33. Valtchev, P., Missaoui, R.: Building Concept (Galois) Lattices from Parts: Generalizing the Incremental Methods. In Proc. 9th Int'l. Conf. on Conceptual Structures: Broadening the Base (ICCS '01), Springer-Verlag, London, UK, pp. 290-303, 2001.
34. Valtchev, P., Missaoui, R., Pierre Lebrun, P.: A Partition-based Approach towards Constructing Galois (Concept) Lattices. Discrete Mathematics 256(3): 801-829, 2002.