Decision Support Systems 52 (2011) 40–51

Contents lists available at ScienceDirect

Decision Support Systems

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

CLAP: Collaborative pattern mining for distributed information systems

Xingquan Zhu [a,b,*], Bin Li [a], Xindong Wu [c,d], Dan He [e], Chengqi Zhang [a]

[a] QCIS Centre, Faculty of Eng. & Info. Technology, Univ. of Technology, Sydney, Ultimo 2007, Australia
[b] Dept. of Computer Science & Eng., Florida Atlantic University, Boca Raton, FL 33431, USA
[c] Dept. of Computer Science, University of Vermont, Burlington VT 05404, USA
[d] School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China
[e] Dept. of Computer Science, Univ. of California at Los Angeles, Los Angeles, CA, 90095, USA

ARTICLE INFO

Article history:
Received 8 October 2010
Received in revised form 4 May 2011
Accepted 15 May 2011
Available online 27 May 2011

Keywords:
Distributed data mining
Distributed association rule mining
Frequent item-sets
Bloom filter

ABSTRACT

The purpose of data mining from distributed information systems is usually threefold: (1) identifying locally significant patterns in individual databases; (2) discovering emerging significant patterns after unifying distributed databases in a single view; and (3) finding patterns which follow special relationships across different data collections. While existing research has significantly advanced the techniques for mining local and global patterns (the first two goals), very little attempt has been made to discover patterns across distributed databases (the third goal). Moreover, no framework currently exists to support the mining of all three types of patterns. This paper proposes solutions to discover patterns from distributed databases. More specifically, we consider pattern mining as a query process where the purpose is to discover patterns from distributed databases with patterns' relationships satisfying user specified query constraints. We argue that existing self-contained mining frameworks are neither efficient, nor feasible to fulfill the objective, mainly because their pattern pruning is single-database oriented. To solve the problem, we advocate a cross-database pruning concept and propose a collaborative pattern (CLAP) mining framework with cross-database pruning mechanisms for distributed pattern mining. In CLAP, distributed databases collaboratively exchange pattern information between sites so that each site can leverage information from other sites to gain cross-database pruning. Experimental results show that CLAP fits a niche position, and demonstrate that CLAP not only outperforms its other peers with significant runtime performance gains, but also helps find patterns incapable of being discovered by others.

1. Introduction

Many applications possess data collected from distributed sources [42,57]. Examples include market basket transaction data from different branches of a wholesale store, insurance claim data from different states, patient health records from different hospitals, census data of different states in one particular year, among many others. Even for one single database, the temporal or spatial relationships may also provide multiple views for the underlying data [11,21]. For example, transaction data [6] collected in a single wholesale store over different time periods can be regarded as multiple correlated databases. Census data of a certain state in different years [26], and patient records of one hospital from different time periods, can also form a collection of multiple databases. For years, knowledge discovery and data mining (also referred to as KDD) have demonstrated themselves to be an effective tool to search for novel and actionable patterns and relationships in the data [42]. Examples of patterns of interests include, but are not limited to, classification models (decision trees or statistical reasoning models) [3,19,45], clusters [14,18], and association rules [2,4,24,33].

From an association rule mining perspective, past research has made significant efforts to discover a variety of patterns, such as frequent item-sets, temporal, spatial, and/or sequential association rules, closed patterns or sequential patterns. Common challenges in this area are usually twofold: (1) identifying patterns from a single (large volume) database [55] or from data with continuous volumes [35,58] (referred to as local patterns or *L-pattern* mining in this paper); and (2) discovering new patterns by unifying multiple databases into a single view [4,57] (referred to as global or *G-pattern* mining in this paper). For distributed databases, a common goal is to discover *G*-patterns, which are trivial in local databases, but significant after multiple databases are unified into a single view. Collective data mining [31] represents the most typical research work in the area. A common practice is to act on local databases, and forward promising local candidates to a central place for synthesizing [12,50].

For distributed databases, *G*-patterns are important because they contain knowledge that is hard, if not impossible, to be realized by *L*-patterns [61]. In practice, there is a third type of pattern that may help discover data relationships across multiple distributed databases.

* Corresponding author at: QCIS Centre, Faculty of Eng. & Info. Technology, Univ. of Technology, Sydney, Ultimo 2007, Australia. Tel.: +1 61 2 9514 1885.
E-mail addresses: xqzhu@cse.fau.edu, xqzhu@it.uts.edu.au (X. Zhu).

0167-9236/$ – see front matter.
doi:10.1016/j.dss.2011.05.002

Take a wholesale store with three branches, *A*, *B*, and *C* as an example where a store manager was organizing data from these three branches for intelligent data analysis, he/she may easily raise concerns such as:

Q_1: What are the patterns frequent in *A*, *B* and *C*? *i.e.*, $(A \geq \alpha)$ & $(B \geq \alpha)$ & $(C \geq \alpha)$, where α is the threshold in finding frequent patterns, and $A \geq \alpha$ means that a pattern's support value in database *A* should be no less than the value α.
Q_2: What are the frequent patterns which appear more often in *A* than in *B*, but infrequent in C? *i.e.*, $(A > B \geq \alpha)$ & $(C < \beta)$
Q_3: What are the patterns whose support of differences in stores A and B are greater or equal to the value α? *i.e.*, $|A\text{-}B| \geq \alpha$.

The above concerns lead to the problem of finding pattern relationships across a number of data collections. This problem is essentially different from local and global pattern mining mainly because users are interested in neither local significant patterns (*i.e.*, *L*-patterns) nor global significant (*i.e.*, *G*-patterns). In reality, when users are exposed to the data collected from multiple databases or multiple data sources,[1] it is natural to refer to a comparative study tool for knowledge and pattern discovery. In addition, it is often the case that users know some basic features of the data, such as the date and time each database was collected, or the region or entity each database may represent. What remains unclear is the relationship of the patterns hidden across the multiple data collections. For example, the store managers may want to find customers' gradually increasing shopping patterns in a certain period of time, or a microbiologist may want to find disease patterns along an evolving order. For these purposes, discovering pattern relationships across multiple databases (referred to as inter-pattern or *I-pattern* mining in this paper) can be a very important part of the KDD process.

The above observations motivate the necessity of finding *I*-patterns from multiple databases, where patterns need to be discovered in individual databases and further compared across different data collections. Although this problem seems easy to solve by simply mining a "seed" database and then comparing patterns across all databases, in practice, *I*-pattern discovery is severely challenged by the following practical issues: (1) databases may be physically distributed so that intensive data transmission across sites should be avoided; (2) due to data privacy or other concerns, data aggregation for mining should be discouraged; and (3) in order to find a pattern *p*'s relationships across multiple databases, one has to scan each database to check *p*'s frequencies with respect to each individual database. This database scanning process is heavily time-consuming if the number of patterns for comparison is large. In addition to the above three issues, if we consider *L*-, *G*-, and *I*-patterns as a whole, we have to face the challenge of devising a unified framework capable of finding all three types of patterns in distributed data environments. Under such circumstances, we believe that the major technique challenges are fourfold:

- System Framework: How to design a unified data mining framework capable of discovering all three types of patterns?
- Mining Procedures: How to transfer a user's mining query into actionable mining activities, so that the mining results from distributed sites can form legitimate answers?
- Data Transmission: What type of information should be exchanged between distributed sites? Also, how should the information be exchanged?
- Cross-database Pruning: How to carry out data mining activities by leveraging information from different sites? In other words, how to enable cross-database pattern pruning so that messages exchanged between sites can speed up the mining process?

[1] In this paper, multiple databases, multi-databases, and multiple data sources are interchangeable terms.

This paper reports our recent progress in resolving the above problems, from both system and algorithm design perspectives. We consider pattern mining as a query process where the purpose is to discover patterns satisfying user specified constraints. To achieve distributed pattern mining, we propose a collaborative pattern mining (CLAP) framework with its own unique method to enable cross-database pruning.

The remainder of the paper is organized as follows. Section 2 reviews existing work in the literature. Section 3 formally defines the problem and discusses pattern queries for mining. Section 4 provides an overview of the distributed pattern mining frameworks. Section 5 articulates technical details of the proposed CLAP mining framework. We report experimental results in Section 6, and conclude in Section 7.

2. Related work

Mining distributed databases [22,30,38]are a practical issue and a large amount of research work has significantly advanced the techniques for distributed classification [3,34,45], clustering [14,18], OLAP [11,21], frequent pattern mining [2,4,12,24], stream data mining [35,46], and database similarity assessments [32,49]. Presumably, nearly every major data mining research area has at least one distributed mining module or algorithm. The main themes of these research activities share striking similarities in the sense that they all intend to unify, and/or compare distributed data sources to achieve a common goal.

From clustering and classification perspectives, the pattern discovery from distributed databases problem arises of how to train global models by leveraging information from multiple databases. This can be achieved by either aggregating data into a single view or integrating models built from single databases [17,36]. Kargupta et. al. [31] proposed a collective data mining framework with a primary key to unify all data into a single view. Similar assumptions were also made for privacy preserving data mining [29,43], cluster ensembling [18], and kernel based model integration [17] for learning heterogeneous data. Yin et. al [53] previously proposed a CrossMiner for classification from multiple relational databases. Wang et. al. [48] addressed the problem of reinforcement clustering of multi-type inter-related objects (*e.g.* web documents). The problem of frequent pattern mining for distributed databases has also been well studied [1,4,12,23,31,33,37,44,50,54,55], where count distribution, data distribution, and candidate distribution are three basic mechanisms [31]. Along with all research activities, the focus has been primarily on mining large volume databases or continuous volume data streams (*i.e.*, mining *L*-patterns), or unifying patterns discovered from single databases into new knowledge (*i.e.*, mining *G*-patterns). Some system architectures also exist to discover frequent patterns from terabyte-scale data-sets running on cluster systems [9], by using compressed data structures (similar to FP-tree [24]) and succinct encoding methods. Such frameworks and solutions, however, typically limit their scope to the data volumes but have no mechanism to comparatively study multiple databases and discover their relationships at pattern levels.

In short, the deficiency of the existing work for distributed database mining is mainly threefold: (1) they lack general cross-database pruning mechanisms; (2) they have no effective message exchanging paradigm but mainly switch patterns in raw formats; and (3) they are not capable of mining all three types of patterns (*L*-, *G*-, and *I*-patterns). In comparison, this paper focuses on finding all types of patterns from distributed databases under a unified mining framework.

When data involve multiple (distributed/centralized) sources, one of the most important tasks is to assess the similarity between databases to discover structural information between databases for clustering [56] or classification [59]. Parthasarathy [39] and Li [32]

have previously addressed the problem of database similarity assessment by comparing association rules from different databases, *e.g.* the identical rules discovered by different databases and the numbers of instances covered by identical rules. The importance of finding differences between databases has been addressed by many researchers [5,15,49,52], and most methods focus on comparing a pair of databases one at a time. Webb et al. [49] proposed a rule based method to explore a contrast set between two databases. Xu et al. [51] proposed to discover comparative opinions between products from customer reviews. In [52], we proposed methods to evaluate the conceptual equivalence between two databases. Ji et al. [27] proposed methods to explore minimal distinguishing subsequence patterns between two data-sets, where the patterns take the form of "frequent in database *A*, but significantly less frequent in database *B*", *i.e.* $\{(A \geq \alpha)\ \&\ (B \leq \beta)\}$. All these methods focus on finding differences (in terms of data items or patterns) between two data-sets, but they cannot support complex queries like the ones in the Introduction. Therefore, this type of work is a sub-set of our framework, and our goal is to address a broader area of problems in pattern discovery from distributed databases.

Research in database queries has made significant efforts in supporting data mining operations [8,28,47,60], with extensions of the database query languages to support mining tasks, but most research effort has focused on a single database with relatively simple query conditions. Two works are closely related to this research: (a) the complex mining optimization system proposed by Jin and Agrawal [28]; and (b) our recent work on relational pattern discovery across multiple databases [60]. In [28], Jin and Agrawal presented an SQL-based mechanism for mining frequent patterns across multiple databases, with the objective of optimizing users' queries to find qualified patterns. The essential difference between work in [28] and the proposed research is twofold: (1) the efforts in [28] only focus on enumerating query plans and choosing the one with the least cost. Instead of optimizing queries our research will propose a distributed data mining framework to support users' queries to find broader types of patterns; (2) because of the limitations of their pattern mining framework (relying on each single database), the solution in [28] can only answer simple queries like $\{(S_i \geq \alpha_1)\ \&\ (S_j \geq \alpha_2)\ \&\ (S_k \leq \beta)\}$, *i.e.*, each element of such a query must explicitly specify one single database and its corresponding threshold value. As a result, their methods cannot answer complex queries like Queries 2 and 3 in the Introduction, and therefore its applicability is limited; and (3) the methods in [28] are only applicable for centralized databases, whereas we intend to mine patterns from distributed databases. In [60], we have proposed a solution to discover relational patterns (*e.g.*, *I*-patterns) across multiple databases, which requires the aggregation of all databases at a central place, which is not feasible for distributed mining scenarios.

In short, although the distributed pattern mining problem has been extensively addressed in the literature, no framework is currently available for mining all three types (*L*-, *G*-, and *I*-) of patterns in distributed scenarios. As the major contribution of this work, we propose a distributed mining framework and a number of algorithms to resolve the key challenges, such as cross-database pruning for distributed mining.

3. Problem definition & query decomposition

Given a number of distributed databases D_i, $i = 1,\ldots, n$, each of which corresponds to an individual site S_i, $i = 1,\ldots, n$, we assume that all distributed sites are able to compute and communicate with others, and a dedicated *master* site is provided for users to submit queries/constraints. The pattern discovery from distributed databases problem is finding patterns complying with the users' queries without aggregating data to a central place (*e.g.*, the master site).

A pattern, *P*, discussed in this paper takes the form as an *item-set*, *i.e.* a set of items satisfying user queries/constraint(s). The *support* of a pattern *P* in a database D_i, represents the ratio between the number of times *P* appears in D_i and the total transactions in D_i.

A user's query/constraint specifies the patterns he/she intend to discover. For example, a user can specify $\{S_i \geq \alpha\}$ to indicate that he/she intends to find patterns from site S_i with all legitimate patterns' support larger than or equal to the threshold α. Assuming that *X* and *Y* denote two databases, we define the following two types of relationship factors and four operators to help users confine their queries.

Relationship and set operators:

- $X \geq \alpha$ ($X > \alpha$) indicates that a pattern's support value in *X* is no less than α (*X* is larger than α).
- $X \leq \alpha$ ($X < \alpha$) indicates that a pattern's support value in *X* is no larger than α (*X* is less than α)
- $X \cup Y$ indicates a virtual set which is the union of the transactions of *X* and *Y*.

Arithmetic operators:

- $X + Y$ indicates the summation of the support in *X* and *Y*
- $X - Y$ indicates the subtraction of the support in *Y* from the support in *X*
- *X* & *Y* indicates the operation of *X* and *Y*
- *X* | *Y* indicates the operation of *X* or *Y*
- |*X*| indicates the absolute support value in *X*.

A user's query is a combination of the above operators for finding patterns from distributed databases. More specifically, a query should involve at least one database and one relationship operator, *e.g.*, $\{S_i \geq \alpha\}$. A query may also involve multiple relationship and arithmetic operators, which is often the case in reality. Following this process, the mining of the *L*-, *G*-, and *I*-patterns can be achieved by using different queries. For instance, the following examples list the queries for each type of pattern:

- *L*-pattern query examples: $Q = \{S_i \geq \alpha\}$ or $Q = \{S_j \geq \alpha\}$
- *G*-pattern query example: $Q = \{(S_i \cup S_j) \geq \alpha\}$
- *I*-pattern query example: $Q = \{S_i \geq S_j \geq \alpha\}$

Due to limitations of the pattern mining process, a user's query cannot take arbitrary forms, but has to involve at least one relationship operator $\geq$ (or $>$) with a numerical threshold value following this operator. For example $Q = \{S_i \geq S_j \geq S_k\}$ is not a valid query; whereas $Q = \{S_i \geq S_j \geq S_k \geq \alpha\}$ is. The reason we require a valid query is because without a threshold α, it is practically infeasible to find all patterns satisfying $Q = \{S_i \geq S_j \geq S_k\}$.

3.1. Query decomposition

A query decomposition process is needed for the following reasons: (1) from the data mining perspective, it is often the case that not all parts of the query comply with the down-closure property [2], *i.e.*, any sub-set of a frequent item-set is also frequent. For example, the "$\leq$" and "$<$" relationship operators normally do not comply with the down closure property. It is obvious that even if a pattern, say {abc}, in S_i, does not satisfy $S_i \leq \beta$, its super-set, say {abcd}, may still comply with $S_i \leq \beta$. Therefore, we must pre-process a user's query and explicitly decompose it into a set of sub-queries complying with the down closure property, so that the mining module can use these sub-queries for candidate pruning; and (2) from a distributed mining perspective, a site may be involved in only a sub-set of the query. Consequently, we need to decompose each user query into a number of sub-queries, each of which only involves necessary sites in the mining process. In this sub-section, we briefly list five properties for query decomposition; other properties [60] are also available but omitted in the paper.

Property 1. *Given a sub-query which contains a relationship operator "$\geq$" or "$>$", if the sub-query has a single database and a threshold value α*

listed as the antecedent and the consequent of the operator "≥" or ">", respectively, this sub-query complies with the down closure property.

Proof. This property is based on the Apriori rule [2] in frequent itemset mining, which states that if a pattern *P*'s support in a database is less than a given threshold α, then any super-sets of *P* (the patterns growing from *P*) will also have their support less than α. Therefore, if a query involves multiple databases, relationship operator "≥" or ">", and a single threshold value α, we may decompose this query into a set of sub-queries with each single database and the threshold value α listed as the antecedent and the consequent the relationship operator, respectively. For example, a query $\{A \geq B \geq C \geq \alpha\}$ can be decomposed into three sub-queries $(A \geq \alpha)$, $(B \geq \alpha)$, and $(C \geq \alpha)$, each of which strictly complies with the Apriori rule. It is obvious that if a pattern *P* violates any one of these three sub-queries, there is no way for *P*, as well as *P*'s super-sets, to be a qualified pattern.

Property 2. *Given a sub-query which contains a relationship operator "≥" or ">", if the sub-query has the sum ("+") of multiple databases and a threshold value α as the antecedent and the consequent of the relationship operator "≥" or ">", respectively, this sub-query complies with the down closure property.*

Proof. Given a pattern *P* and any of its sub-patterns *Q*, assuming *P's* and *Q*'s supports in *A*, *B* and *C* are p_1, p_2, p_3 and q_1, q_2, q_3 respectively, it is obvious that $q_1 \geq p_1$, $q_2 \geq p_2$, $q_3 \geq p_3$. If $(p_1 + p_2 + p_3) \geq \alpha$, then it is obvious that $(q_1 + q_2 + q_3) \geq (p_1 + p_2 + p_3) \geq \alpha$. Therefore, the property 2 is true. This property states that if a sub-query sums up multiple databases and is followed by factors "≥" or ">" and a threshold value α, then the sub-query strictly follows the down closure property and can be directly used for pattern pruning.

Property 3. *Given a sub-query which contains a relationship operator "≥" or ">", if the sub-query has the support difference of two databases, say $(S_i - S_j)$, and a threshold value α listed as the antecedent and the consequent of the relationship operator "≥" or ">", respectively, this sub-query can be further transformed into a sub-query like $S_i \geq \alpha$, which still complies with the down closure property.*

Proof. It is obvious that if $(A - B) \geq \alpha$, then $A \geq (B + \alpha)$. Since a pattern's support in a database cannot be negative, we have $A \geq \alpha$.

Property 4. *Given a sub-query which contains a relationship operator "≥" or ">", if the sub-query has the absolute support difference of two databases, say $|S_i - S_j|$, and a threshold value α listed as antecedent and the consequent of the relationship operator "≥" or ">", respectively, this query can be transformed into a sub-query like $\{(S_i \geq \alpha) \mid (S_j \geq \alpha)\}$, which still complies with the down closure property.*

Proof. It is obvious that if $|A - B| \geq \alpha$, then we have $(A - B) \geq \alpha$ or $(A - B) \leq -\alpha$, which are equivalent to the inequations $A \geq (B + \alpha)$ or $B \geq (A + \alpha)$, *i.e.* $\{(A \geq \alpha) \mid (B \geq \alpha)\}$. For any pattern *P*, if its supports in *A* and *B* are both less than α, there is no way for *P's* super-set to have a higher support than α. Therefore, the pattern *P* still complies with the down closure property.

Property 5. *A sub-query containing relationship factors "≤" or "<"complies with the down closure property.*

Proof. It is obvious that even if a pattern, say $P_1 = \{abc\}$ in a database S_i, does not satisfy $S_i \leq \beta$, its super-set, say $P_2 = \{abcd\}$, may still comply with $S_i \leq \beta$. Therefore, any pattern that does not satisfy query $S_i \leq \beta$ cannot be pruned out, because it can later grow into a longer length pattern, which eventually will satisfy the query constraint $(S_i \leq \beta)$.

In our design, a query is decomposed at the master site based on the above properties. The decomposed sub-queries (which comply with the down-closure property) are placed into Down Closure (DC) sub-sets and are further dispatched to distributed sites. The original query is also kept to validate patterns at the final stage.

4. Pattern mining frameworks

From a system perspective, the problem of distributed mining (for L-, G-, and I-pattern discovery) can be solved by three frameworks: (1) SeQuentiaL Pattern mining (SQLP); (2) PAralleL Pattern mining (PALP); and (3) CoLlAborative Pattern mining (CLAP). The conceptual views of the three frameworks are shown in Fig. 1, where a master node collects user queries and collects mining results from distributed databases DB_1, DB_2,..., DB_n.

In Fig. 1, SQLP and PALP are *self-contained* mining frameworks, because mining is essentially carried out in individual sites without involving data from other sources. For SQLP, pattern mining is initialized at a seed database (*i.e.*, DB_1 in Fig. 1(a)) with results passed on to the second database for verification. The above process repeats until patterns are verified by all databases involved in the query. For example, to answer $Q_2 = \{(A > B \geq \alpha)\ \&\ (C < \beta)\}$ in Section 1, *SQLP* may start from database *A* to find frequent patterns satisfying $\{A > \alpha\}$, and then pass on patterns to database *B* to find patterns satisfying $\{A > B\}$. Any patterns not satisfying the query will be pruned out immediately.

Instead of mining and verifying patterns in a sequential way, *PALP* carries out the mining of individual databases in parallel, and collects all patterns in a central location to find the ones satisfying the user queries. In Fig. 1(b), the mining is initiated in all databases, and the answers are forwarded to the master site for validity check. For example, to answer $Q_2 = \{(A > B \geq \alpha)\ \&\ (C < \beta)\}$ in Section 1, *PALP* concurrently discovers patterns from each single database (*A* and *B*), and then collects all patterns to find those that are qualified. One should be aware that it is technically not feasible to find patterns which satisfy $\{C < \beta\}$ by using database *C* alone, because no deterministic pruning rules will hold and one has to list all the candidates, if he/she intends to do so. Therefore, *PALP* will concurrently mine patterns from *A* and *B*, and then pass on the patterns to *C* for verification.

For both SQLP and PALP, the pattern mining process (candidate generation and pruning) is carried out at each single site. The inherent disadvantage of such self-contained mining frameworks is that pattern generation and pruning are essentially single-database-oriented and inefficient for distributed mining. Taking a simple query like $Q = \{(S_i \geq \alpha)\ \&\ (S_j \geq \alpha)\}$ as an example, for small α values, a large number of patterns may satisfy either $S_i \geq \alpha$ or $S_j \geq \alpha$, but very few of them satisfy $(S_i \geq \alpha)\ \&\ (S_j \geq \alpha)$. Consequently, a pruning process utilizing information from S_i and S_j is much more efficient than mining S_i and S_j alone.

Different from self-contained mining where sites are independent of each other and the mining process is limited to the local data, joint mining intends to let distributed databases collaborate with each other for pattern discovery. Ideally, a joint mining framework should meet the following three criteria for pattern discovery: (1) being able to unify distributed databases for cross-database pattern pruning; (2) being able to answer complex queries for mining all three types of (L-, G-, and I-) patterns; and (3) being able to scale up to large volume databases with limited bandwidth consumption and no source data sharing.

Fig. 1(c) proposes a framework, CLAP, which carries out mining activities in a "joint" manner. CLAP allows the distributed sites to communicate with each other and exchange messages, so the mining is carried out at distributed sites without any data integration. The cross-database pattern pruning is achieved by using messages exchanged between sites.

Several concerns remain regarding the efficiency of the proposed collective and collaborative mining frameworks:

- What type of information should be exchanged between sites for effective mining and data privacy protection?

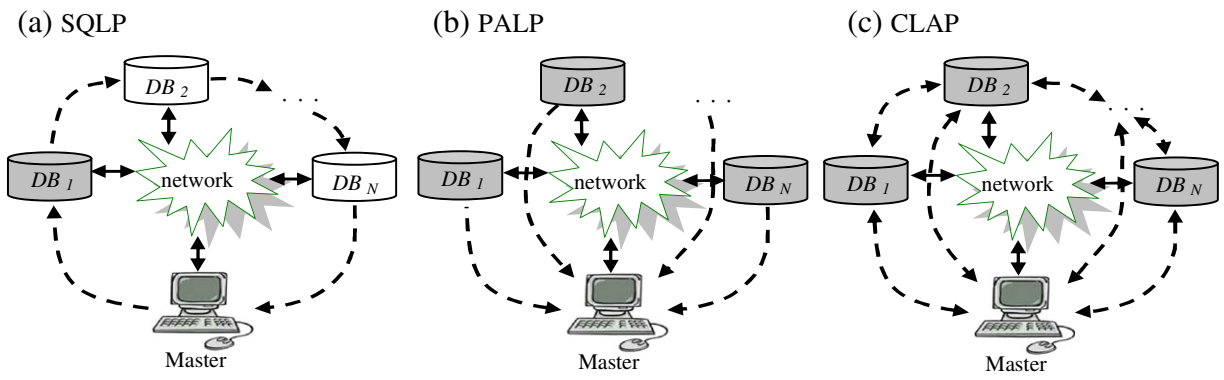


Fig. 1. Conceptual views of the data and knowledge flow of different mining frameworks: solid lines indicate physical connections and dash lines show the data and knowledge flow. Gray nodes indicate nodes actually carrying out the mining activities (candidate generation and pruning). In SQLP, patterns are discovered from one DB and sequentially passed to others for verification; in PALP, patterns are generated in each single database and forwarded to the master site for validation; and in CLAP, the mining activities are carried out in distributed sites with messages exchanged between sites for cross-database pruning.

- How to exchange messages between sites for effective transmission and mining?
- How to utilize information from other sites to fulfill cross-database pattern pruning and mining?

This paper proposes to rely on the exchanging of pattern filters between distributed sites for cross database pruning. More specifically, the distributed sites will exchange the complete set of length-l patterns with other sites for cross database pruning, so each site can immediately prune out candidates which do not satisfy the query. Because exchanging patterns and checking pattern existences in a database are time-consuming, we will employ bloom filters to accelerate the whole mining process.

5. Clap: Collaborative pattern mining

Collaborative pattern mining advocates pattern discovery in a distributed manner with each distributed site carrying out pattern pruning in collaboration with its peers, by employing the Bloom Filter (BF) [7,10,13,16,20] based pattern switching mechanism. In following sub-sections, we first briefly introduce the bloom filter and its potential for distributed mining. In Section 5.2 we introduce a depth-limited FP-growth process which utilizes bloom filters to achieve cross-database pruning. The collaborative pattern mining framework is introduced in Section 5.3.

5.1. Bloom filters for distributed mining

A bloom filter (BF) is a space efficient data structure, which consists of k hash functions, $H_1(\cdot)$, $H_2(\cdot)$, ..., $H_k(\cdot)$, and an m bits binary array. The strength of a BF tests whether a given element is a member of a set in a very effective way [7,16,20]. Fig. 2 shows given elements $x_1, x_2, \ldots, x_n$, each of which is hashed by k hash functions to k locations of the m-bits array. The m-bits array is initially set to 0, but a bit j of the array is flipped to 1, if any hash function maps a pattern x_i to the jth location. Following the above procedure, one can add all n patterns $x_1, x_2, \ldots, x_n$ into the bloom filter. To check whether a pattern x_t exists in a bloom filter or not, one can use all k hash functions to map x_t to k positions. If any of the k positions is 0, x_t does not exist in the bloom filter. If all k bits are 1, we conclude that x_t exists in the bloom filter with regard to a false positive rate (the bits may be set to 1 during the insertion of other patterns rather than x_t).

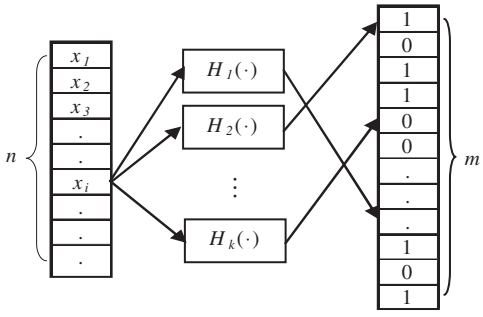


Fig. 2. Bloom filter architecture.

Assume the size of the bloom filter array is m bits, the probability that a certain bit is not set to one by a certain hash function $h(\cdot)$ during the insertion of an element is

$$1-\frac{1}{m} \tag{1}$$

Given k hash functions $h_1(\cdot)$, $h_2(\cdot)$,..,$h_k(\cdot)$, the probability that a certain bit is not set to one by any of the k hash functions is given below

$$\left(1-\frac{1}{m}\right)^k \tag{2}$$

Because the insertion of each element is independent, after inserting n elements to the bloom filter, the probability that a certain bit is set to 1 is given in Eq. (3)

$$1-\left(1-\frac{1}{m}\right)^{nk} \tag{3}$$

Assume an element x was not inserted into the bloom filter earlier, a false positive happens only if all of the k hash positions of x are set to 1. This is equivalent to the probability shown in Eq. (4), which asserts that the false positive rate of a bloom filter decreases as the filter size (m value) increases, and increases as the number of inserted items (n value) increase.

$$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^k \approx \left(1-e^{-kn/m}\right)^k \tag{4}$$

Assume two sites S_i and S_j are carrying out pattern mining to discover patterns frequent in both S_i and S_j, bloom filters can help both sites achieve cross-database pruning by switching their bloom filters BF_i and BF_j (each contains patterns frequent at one site). The

employment of the bloom filters has a number of advantages. First, a bloom filter is fast for membership checks. Assume site S_i has the bloom filter BF_j from S_j, S_i can query BF_j, with $O(1)$ time complexity, to check whether a pattern exists in S_j or not. Secondly, a bloom filter is space efficient. Exchanging bloom filters between sites is much more efficient than exchanging patterns between sites. Thirdly, a bloom filter's false negative value is zero. In other words, if S_i queries BF_j and finds that a pattern x does not exist in BF_j, then x is indeed not frequent in S_j. As a result, S_i may safely remove x. So the cross-database pattern pruning can be achieved.

5.2. Depth-limited pattern growth for cross-database pruning

By using bloom filters, a naive cross-database pruning approach, following the Apriori principle [2], can be implemented as follows:

1. Given a site S_i, use the Apriori mining approach to generate a complete set of length-l patterns.
2. Use frequent length-l patterns in site S_i to construct a bloom filter (BF_i-l), and broadcast BF_i-l to other sites.
3. After site S_i receives the bloom filters BF_j-l from other sites, it can query BF_j-l and prune out length-l patterns in S_i and then grow length-$(l+1)$ patterns.
4. Set $l \leftarrow l+1$ and repeat Steps 2 to 4 until no more frequent patterns can be discovered from any sites.

The main disadvantage of the above cross-database pruning approach is that it critically relies on the Apriori principle, where repetitive database scanning is heavily time-consuming and will significantly slow down the mining process. In this sub-section, we propose a new depth-limited FP-growth (DLFP-growth) which combines the strength of the bloom filter and FP-growth for distributed sites to achieve cross-database pruning.

Intuitively, although FP-growth is effective for pattern mining, it is, however, unsuitable for cross-database pruning. This is because FP-growth is a depth-first recursive process which starts from an item "*a*" and discovers all patterns related to "*a*" before it moves on to the next item "*b*". Such a depth-first approach makes the collection of the complete set of length-l patterns unavailable until the whole algorithm ceases. In other words, we cannot collect all length-l patterns from site S_i and distribute them to other sites until the whole mining process at S_i stops (then pattern switching between sites becomes meaningless). Alternatively, we can forbid the recursive FP-growth process from going deeper once the length of the pattern reaches a limit l, then force FP-growth to turn to the next items and continue to discover the complete set of length-l patterns (*i.e.*, turn the depth-first search into a depth-limited approach).

The depth-limited FP-growth (DLFP-growth), as shown in Fig. 3, takes four parameters, an FP tree, a base set BS, a length constraint l, and a set of bloom filters, if they exist, *BF*[], as input. On Step 1, DLFP-Growth will terminate and stop growing the pattern longer if the length of the pattern (enclosed in the BS) reaches the length l.

In Step 2, the pattern growth will be carried out for each item x_i of the given FP tree. This process utilizes the bloom filters collected from distributed sites for cross-database pruning. Given a base set BS and item x_i, the new pattern ϑ for growth is the concatenation of BS and x_i, as shown in Step 2.a. Instead of directly building an FP tree for x_i, which is a relatively expensive process, we can query bloom filters *BF*[] and prune out x_i if any sub-sets of ϑ do not exist in *BF*[]. For example, assume $BS = \{abd\}$ and $x_i = g$, then the pattern under growth is $\vartheta = \{abdg\}$. Assume a bloom filter in *BF*[], denoted by BF_j-3, contains length-3 patterns from site S_j. We query any length-3 sub-sets of ϑ, such as $\vartheta_1 = \{bdg\}$, from BF_j-3. If ϑ_1 does not exist in BF_j-3, we can safely prune out $x_i = g$ without growing an FP tree for x_i because pattern ϑ is not frequent in the distributed sites S_j, so there is no need to grow it in the local site S_i.

DLFP-Growth (*FP-tree*, *BS*, *l*, *BF*[])
Input: (1) An FP tree: *FP-tree*; (2) A Base set *BS*; (3) a length constraint *l*
(4) Bloom filters collected from other sites, if exist: *BF*[]
Output: Length *l* pattern set *P*

1. **If** *BS* contains *l* items
   a. Return BS
2. **For** each item x_i in the header table of *FP-tree*
   a. $\vartheta \leftarrow BS \cup x_i$
   b. **If** (*BF*[] exist and any subset of ϑ does not exit in BF[])
      i. **Continue** //Prune out a_i and move to next items
      EndIf
   c. Build an $FP\text{-}tree_i$ from *FP-tree* (please refer to [11] for details)
   d. $p \leftarrow$LCFP-Growth(FP-tree$_i$, ϑ, *l*, *BF*[])
   e. $P \leftarrow P \cup p$
   EndFor
3. Return (P)

Fig. 3. Depth-Limited FP-growth process.

It is worth noting that the DLFP-growth process can be easily adjusted to fit different situations through the tuning of the parameters l and BF[]. For example, if we set $l = -1$ and BF[] = null, then DLFP-growth degenerates as the traditional FP-growth. On the other hand, setting l to any values greater than 0 with BF[] = null, one can collect all length-l patterns without utilizing any bloom filters from other sites. In the next sub-section, we will articulate technical details of using DLFP-growth for collaborative pattern mining from distributed databases.

5.3. Collaborative pattern mining with DLFP-growth

In Fig. 4, we list major steps for a site to carry out collaborative pattern mining using length constrained FP-Growth, where cross-database pruning is achieved through the following three major steps:

***CLAP*: Collaborative Pattern Mining (S_i, *SQ*, *l*)**
Input S_i: A distributed site along with its data *D*
SQ: A subquery delivered from the master site
l: The pattern length constraint for cross site pruning through BF.
Output Φ: pattern set satisfying the subquery *SQ*

1. $\Phi \leftarrow \emptyset$
2. $FP_i \leftarrow$ build_FP_Tree (S_i, *SQ*)
3. length_*l*_set ← DLFP-Growth (FP_i, *null*, *l* ,*null*)
4. $BF\_l \leftarrow$ build_BF (length_*l*_set) // a BF contains length *l* patterns
5. *S*[]← check relevant sites (*SQ*) // check sites relevant to the subquery *SQ*
6. Request length_*l*_set from sites *S*[] //request length *l* patterns from relevant sites
7. **Switch** (*Events*) // event actions
8. **Case:** receiving length_*l*_set request from sites S_j // a site request patterns
9. Send BF_i_l to S_i if available
10. **Case:** site S_j responds BF_*l* request //a site confirm request sent on Step 6
11. BF_*l*[*j*] ← receive BF_*l* from S_j
12. **Else:** **For** every item a_i in FP_i //FP-Growth with cross site pruning
13. P ← DLFP-Growth (FP_i, *null*, -1, BF_*l*[])
14. Φ ← Φ ∪ P
15. **EndFor**
16. Send Φ to the master site
17. **EndSwitch**
18. **Return** (Φ)

Fig. 4. Collaborative pattern mining framework.

- A local site S_i generates the complete set of length-l frequent patterns by calling DLFP-growth. (Step 3)
- Site S_i constructs a bloom filter, BF_i_l, by using length-l frequent patterns discovered at Step 3, and sends BF_i_l to distributed sites. (Steps 4, 8, and 9)
- Site S_i carries out pattern growth with cross-database pruning, by using $BF\_l[]$ collected from other sites. (Steps 12, 13, and 14)

The framework in Fig. 4 is essentially an asynchronous distributed mining module, which means that each distributed site can work independently without synchronizing with any other sites. For any site S_i, a sub-query SQ is accepted from the master site as an input, and then the sites relevant to the sub-query SQ are determined (Step 5). After that, S_i will send a request to each of the relevant sites and ask them to send a bloom filter containing length-l patterns to S_i. The mining process then runs into an event driving loop between Steps 7 and 17. More specifically, if site S_i receives a request from site S_j, which is asking for length-l patterns, S_i will immediately send BF_i_l to S_j as shown in Steps 8 and 9. If a site S_j responds to S_i's request at Step 6, S_i will collect the bloom filter from S_j and include it with the bloom filter arrays $BF[]$. Under any other circumstances, S_i will continuously grow patterns by using the bloom filters collected from other sites (Steps 12 to 15).

After each site completes the mining process, the results (patterns and their actual support values) are delivered to the master site, which will further verify and finalize valid patterns. For example, for a query like $Q=\{(S_i \geq S_j \geq \alpha)$ and $S_k \leq \beta\}$, the master site needs to collect patterns satisfying $(S_i \geq S_j \geq \alpha)$ and then deliver the pattern to S_k to finalize those with their support values less or equal to β.

Alert readers may have noticed that a large portion of length-l patterns discovered at Step 3 will be re-discovered at Step 13. This raises a concern regarding the extra cost involved at Step 3, especially if this step takes a significant amount of system runtime. In Section 6.2, we will show that when using small l values (*e.g.*, $l=2$ or 3), Step 3 only costs 1–2% (or less) of the runtime compared to the FP-Growth without length constraints. Given that the goal of Step 3 is to enable the cross-database pruning, the extra cost added to this step is of little concern.

Notice that a bloom filter cannot encode support values of the patterns, indicating that CLAP may not directly answer a summation based query like $Q=\{(S_i+S_j)\geq\alpha\}$ because, without knowing the support values of a pattern p in both S_i and S_j, we cannot determine whether p (along with its successors) can satisfy the query $Q=\{(S_i+S_j)\geq\alpha\}$. In addition, even if a pattern p's support in S_i is 0, it may have the summation S_i+S_j greater than α, which makes p a legitimate pattern with regard to the query Q (but mining patterns satisfying $Q=\{S_i\geq 0\}$ are technically infeasible). CLAP solves this problem by repetitively exchanging length-1, length-2 and length-3 item-sets between sites to collect a reasonable set of length-3 patterns from which the pattern growth becomes possible. More specifically, given query $Q=\{(S_i+S_j)\geq\alpha\}$, we first collect length-1 item-sets and their support values for both S_i and S_j, and we exchange item-sets and their values between S_i and S_j, so each site knows exactly the support values of each item in the other site. According to Property 2 in Section 3, any item with its support value $(S_i+S_j)<\alpha$ cannot grow patterns satisfying $Q=\{(S_i+S_j)\geq\alpha\}$. As a result, sites S_i and S_j can prune out length-1 item-sets, and grow and exchange length-2 item-sets between each other. The above process involves a heavy communication cost, so we repeat this process for only a limited number of times ($l=3$ in our experiments), then we let S_i and S_j independently grow without further communication.

5.4. Distributed pattern mining framework comparisons

In Table 1, we briefly summarize the strength and weakness of the three distributed mining frameworks (SQLP, PALP, and CLAP), from system design, functionalities, and data privacy perspectives. The detailed performance comparisons are reported in Section 6. Between all three frameworks, CLAP is the only one with cross-database pruning that is capable of mining all three (*L*-, *G*-, and *I*-) types of patterns.

Table 1
A simple comparison between three distributed mining frameworks. "+" indicates that a framework is positive with respect to the assessment criterion, "−" means negative, "~" represents partially positive (*i.e.*, a framework may partially meet the criterion), and "/" means the criterion is meaningless for that particular framework.

Assessment criteria	SQLP	PALP	CLAP
Mining *L*-patterns?	+	+	+
Mining *G*-patterns?	−	−	+
Mining *I*-patterns?	~	~	+
Distributed mining activities?	+	+	+
Cross-database pruning?	−	−	+
Distributed data structure?	+	+	+
Low memory consumption?	+	~	+
Limited # of DB scanning?	−	+	+
Data privacy concerns?	+	+	+
Effective message switching?	−	/	+
Parallel mining activities?	−	+	+

From a message exchanging perspective, CLAP and SQLP are the only two frameworks requiring message switching between sites (excluding the master site). Comparing CLAP and SQLP, CLAP employs the bloom filter for message switching, whereas SQLP directly passes on the original patterns from one database to another. As a result, CLAP is much more efficient in terms of message switching.

6. Experiments

6.1. Experimental settings

6.1.1. Methods

For comparison purposes, we implement all three frameworks discussed in Section 4. All programs are written in C++ (Borland C++ Builder 6.0). For CLAP, we use open bloom filter [40] as the basis and implement our own bloom filter. In the experiments, the size of the bloom filter (m) is selected so that the ratio between the filter size (m) and the item number (n) is 8, and the number of hash functions is set to $k=5$,[2] which gives a theoretical false positive rate of about 2.14%. For SQLP and PALP, each site uses an FP tree to achieve maximal mining speeds (we implement the FP tree using an STL-like C++ tree class [41]).

6.1.2. Data

Our test-bed, listed in Table 1, consists of two groups of synthetic data-sets generated from an IBM quest data generator [25]. The explanation of the data description used in Table 2 is as follows. T1000k.N10kS1kL20 means a database with one million transactions, 10,000 unique items (N), and 1000 significant patterns (S), where the average length of the maximum length pattern is 20 (L) (L20 + 19 means the combinations of setting L to 20 and 19; more details follow).

The two groups in Table 2 simulate "strong dense (SD)" and "weak sparse (WS)" distributed databases. More specifically, "dense *vs.* sparse" means the number of unique items in the database, and a dense database has a smaller number of unique items compared to a sparse database; "strong *vs.* weak" indicates the similarity or correlations between databases, where "strong" means that distributed databases have high similarities and strong correlations with each other. In Table 3, we report the pair-wise similarities of the WS and SD databases, where each similarity value between row (D_r) and column databases (D_c) is given in Eq. (5). In short, for a specific

[2] For given m and n values, the hash function number (k) that minimizes the probability of the false positives is about $k=0.7\times(m/n)$.

Table 2
Benchmark database characteristics.

Database		Database description
Strong dense databases	SD_1	T1000k.N1kS1000L20
	SD_2	T500k.N1kS1000L20
	SD_3	T250k.N1kS1000 L20
	SD_4	T125k.N1kS1000 L20
Weak sparse databases	WS_1	T1000k.N10kS1000 L20 + 20
	WS_2	T500k.N10kS1000 L20 + 19
	WS_3	T250k.N10kS1000 L20 + 18
	WS_4	T125k.N10kS1000 L20 + 17

support value α, the pair-wise similarity between D_r and D_c in Eq. (5) is calculated as the percentage of the number of rules discovered by both D_c and D_r, in comparison with the total number of rules discovered by D_r. The pair-wise similarity is asymmetric so $DB_\alpha(r,c) \neq DB_\alpha(c,r)$. For the IBM quest data generator, the L value will determine the pattern distributions. Varying the L values will generate databases with very little correlation (whereas fixing the L value will output strongly correlated databases). Therefore, in our experiments, the WS databases are generated by a mixture of two L values.

The values in Tables 3 show that SD databases have very high similarity, *e.g.*, almost all rules discovered in SD_4 are discovered by SD_1 as well, whereas a very small percentage of rules in the WS databases are identical to each other.

$$DB_\alpha(r,c) = \frac{|D_r \cap D_c|}{|D_r|} \tag{5}$$

6.1.3. Measures

The experiments select a number of queries (listed in Table 4) as benchmarks which are provided to a dedicated master site. The queries are further decomposed into a number of sub-queries and are dispatched to corresponding sites if necessary. Although it is possible to re-use previously discovered results to answer a query (*e.g.*, results from $\{S_i \geq 0.5\%\}$ can be re-used by query $\{S_i \geq 0.8\%\}$), for fairness of the comparison, all queries are answered by reinitializing the whole mining process. All algorithms are compared based on their *runtime* performances and/or the size of messages exchanged between sites. The runtime of the systems crucially relies on the underlying queries. For an objective assessment, we define four queries, as shown in Table 4, and will demonstrate the average system runtime performances to answer these queries.

The performance of CLAP relies on two important factors: (1) depth-limited pattern growth for cross-database pruning; and (2) bloom filter based message exchanging between sites. The following sections study CLAP in detail. A comparative study across all three frameworks (including SQLP and PALP) is reported in Sections 6.4.

6.2. Depth limited pattern growth results

As shown in Fig. 4, the cross-database pruning of CLAP relies on the exchange of the length-l patterns between sites. This raises an important issue of finding the proper l value for DLFP-growth to find the complete set of length-l patterns (Step 3 in Fig. 4). Practically,

Table 3
Pair-wise database similarities ($\alpha = 0.5\%$).

(a) Strong dense databases					(b) Weak sparse databases				
DB	SD_1	SD_2	SD_3	SD_4	DB	WS_1	WS_2	WS_3	WS_4
SD_1	1.0	0.91	0.98	0.22	WS_1	1.0	0.04	0.04	0.03
SD_2	0.97	1.0	0.96	0.21	WS_2	0.02	1.0	0.02	0.02
SD_3	0.91	0.84	1.0	0.19	WS_3	0.01	0.01	1.0	0.01
SD_4	0.99	0.92	0.90	1.0	WS_4	0.02	0.02	0.02	1.0

Table 4
Query plan description.

Query	Query constraints
Q_1 (L-pattern)	$\{(S_1 \mid S_2 \mid S_3 \mid S_4) \geq \alpha\}$
Q_2 (G-pattern)	$\{(S_1 \cup S_2 \cup S_3) \geq \alpha\}$
Q_3 (I-pattern)	$\{(S_1 + S_2) \geq \alpha \ \& \ (S_3 \leq S_4 \leq \beta)\}$
Q_4 (I-pattern)	$\{S_1 \geq S_2 \geq S_3 \geq S_4 \geq \alpha\}$

although exchanging length-l patterns between sites can enable cross-database pruning, the process of finding the complete set of length-l patterns adds extra cost to individual sites. So l values must be properly determined to ensure balanced performance gains.

In Fig. 5, we report the ratios between the runtimes of DLFP-growth with different l values and DLFP-growth without any length constraint, as defined in Eq. (6), which shows the extra cost of CLAP in finding the complete set of length-l patterns (compared to the total mining cost for each individual site).

$$r = \text{DLFP}(FP\ tree, null,\ l, null)/\text{DLFP}(FP\ tree,\ null, -1, null) \tag{6}$$

The results in Fig. 5 indicate that for both databases (SD_1, WS_1), the major cost of the FP-growth is the discovery of medium size patterns. For example, for SD_1 the cost of finding all length-3 (and length-2) patterns by DLFP is only 3.7% of the cost of finding all frequent patterns, whereas finding all length-10 (and shorter) patterns takes about 53% of system runtime. This is easy to understand because when l is small, the number of length-l item-sets is only a small portion of the candidate patterns evaluated by the system. Similarly, only a very small portion of patterns in the database have a long length, and the majority of patterns (or candidates) are medium length, which explains why two curves in Fig. 5 are sigmoid in shape.

In order to study the impact of the length-l patterns for cross-database pruning, we choose databases WS_1 and WS_2 and run CLAP mining at WS_1 by using bloom filters from WS_2 with different lengths of patterns, *i.e.* BF_2_l ($l = 2,3,..6$). If mining were carried out on WS_1 alone without using any bloom filters from WS_2, it takes WS_1 21.31 s for tree pruning and eventually outputs 49,660 patterns. The pruning efficiency in this case is 0%. By including bloom filters from WS_2 to assist cross-database pruning, as shown in Fig. 6, we can find that CLAP significantly improves its mining efficiency. For example, when including a length-2 pattern bloom filter (BF_2_2), the tree pruning time for WS_1 is reduced to 2.47 s, and the total runtime (including bloom filter construction) is about 21% of the stand-alone pruning time of WS_1. As pattern length l grows, the time percentage will gradually increase, mainly because mining of the complete set of length-l patterns demands more time (Fig. 6) and the cross-database pruning at WS_1 will have to validate more candidates.

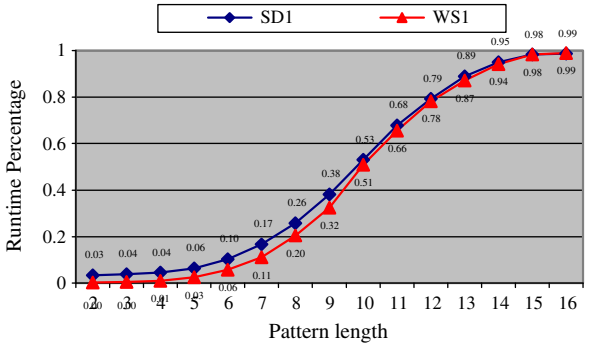


Fig. 5. System runtime for finding different length-l patterns. The x-axis denotes the pattern length l, and the y-axis denotes the percentages of the system runtimes between finding all patterns with length less or equal to l and finding all patterns ($\alpha = 0.7\%$).

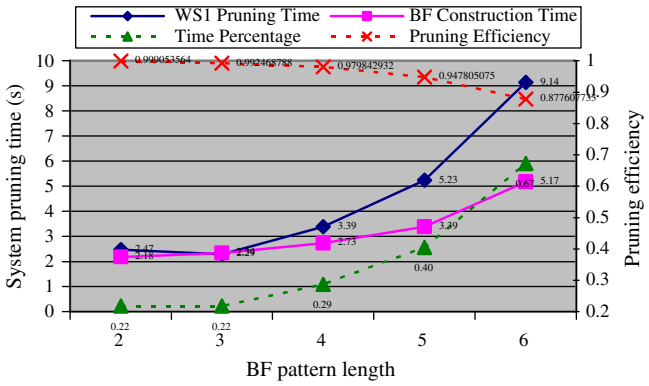


Fig. 6. The CLAP mining results on WS_1 by using bloom filters with different pattern lengths BF_2_l ($l=2,3,..6$) from WS_2. "WS_1 Pruning Time" denotes CLAP pruning time (Steps 12 to 16 in Fig. 4) on WS_1*w.r.t.* different bloom filter pattern lengths. "BF Construction Time" is the bloom filter construction time at WS_2. "Time Percentage" denotes the ratio between the summation of "WS_1 Pruning Time", "BF Construction Time", and "WS_1 Pruning Time", and the runtime of CLAP on WS_1 without using any bloom filters from WS_2 (21.31 s). "Pruning Efficiency" is the ratio between the number of pruned patterns (due to the inclusion of the bloom filters) and the number of patterns without including any bloom filters (49660). The support value is $\alpha=0.5\%$, and solid lines correspond to the left *y*-axis and dash lines correspond to the right *y*-axis.

Interestingly, the results in Fig. 6 show that, although CLAP's cross-database pruning efficiency remains relatively stable for different l values, overall the larger the l values, the less effectively CLAP prunes out irrelevant patterns. We believe that this is mainly because length-l patterns can only help prune length-$(l+1)$ patterns, but not length-$(l$-1) patterns. As the length l grows, patterns with length less than l become a significant portion of the pattern space, but they are not pruned by CLAP. Considering the above factors we set length l to 3 in our experiments.

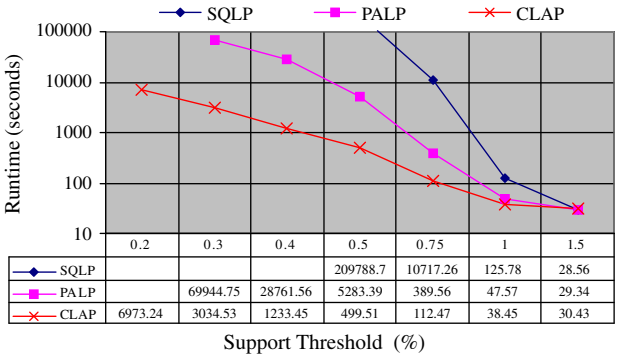
6.3. Bloom filters based inter-sites message exchanging results

Table 5 reports the results of bloom filters built from SD_1 with respect to different threshold values α (the results from other databases are more or less similar to the results in Table 5), where the actual False Positive (FP) rate (the last column) was collected by an average of 10,000 random queries. The results in Table 4 assert that the construction and query of the bloom filters are very efficient, and the bloom filter construction time is only a tiny portion of the tree pruning time. The query time of the bloom filters, which is independent of the filter size, is also efficient and can be achieved in 0.05 s for 10,000 queries. The sizes of the bloom filters are typically several hundred kilo-bytes or less, even for a very small support value (*e.g.*, $\alpha\leq 0.1\%$). Consequently, the exchanging (delivering) of the bloom filters between sites incurs very little extra cost.

In short, the observations in this sub-section conclude that the collection of the complete set of length-l patterns and the construction of the bloom filters add little extra cost to the system. As a result, the employment of the depth-limited pattern growth and bloom filters ensures CLAP can effectively carry out cross-database pruning in a distributed manner.

Table 5
The results of bloom filters for message exchanging between sites ($l=3$) (SD_1 database).

Support α (%)	# of patterns	BF size (K bytes)	BF const. time (s)	BF query time (s/10,000)	Actual FP rate (%)
0.5	23,668	24	0.20	0.051	2.13
0.4	51,955	51	0.48	0.050	2.30
0.3	97,094	95	0.89	0.039	2.34
0.2	175,433	172	1.64	0.040	2.10
0.1	419,898	410	3.98	0.047	2.09



	0.2	0.3	0.4	0.5	0.75	1	1.5
SQLP				209788.7	10717.26	125.78	28.56
PALP		69944.75	28761.56	5283.39	389.56	47.57	29.34
CLAP	6973.24	3034.53	1233.45	499.51	112.47	38.45	30.43

Fig. 7. Query runtime comparison on Q_4 in Table 4 (SD databases).

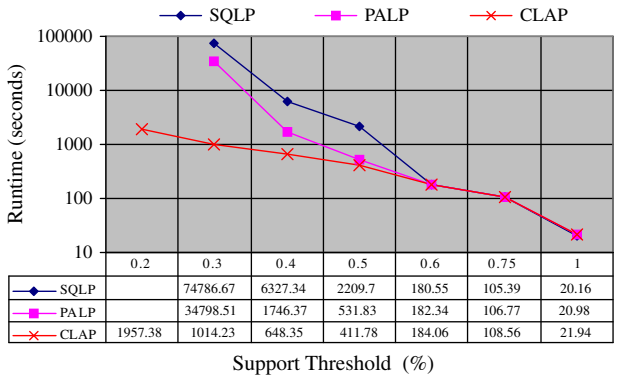
6.4. Comparative studies

Figs. 7 and 8 report the system runtime comparisons across all three frameworks (SQLP, PALP, and CLAP) in answering an *I*-pattern (query Q_4) listed in Table 4. The results are collected with respect to different support values (α). The general setting of the experiments are as follows. For SQLP, mining is invoked at site S_1, with the results sequentially passed on to sites S_2, S_3, and S_4 for validation. For PALP, mining is invoked at all sites simultaneously, and the master site collects and finalizes the patterns satisfying the query (both SQLP and PALP use an FP tree to gain maximum speed). For CLAP, each site uses $l=3$, $m/n=8$, and $k=5$ for depth-limited pattern growth and bloom filter construction. For comparison purposes, we decompose the system runtime of each framework into a number of major components, and report the decomposed runtime in Tables 6.1 to 6.3 to enable the detailed study and comparison of the three frameworks

Between the three frameworks SQLP has the smallest overhead for large support values (*e.g.*, $\alpha\geq 1\%$) because it initiates mining at a seed site and sequentially passes on the mining results to other databases for verification. For large α values, only a very limited number of patterns are discovered from the seed site, so SQLP is quite efficient in answering this type of query. The results in Figs. 7 and 8 support the hypothesis and show that when the value of α is around 1.0%, the runtime performance of all three frameworks are close to each other.

For self-contained mining frameworks, when support values α decrease, the performance of both SQLP and PALP deteriorates dramatically for two reasons. First, the mining activities of SQLP and PALP are single database oriented and as the support value decreases, pruning of the individual FP tree becomes ineffective and time consuming. Secondly, as the support value decreases, the number of patterns satisfying $S_i\geq\alpha$ for each site S_i increases exponentially. For SQLP, each pattern needs to be forwarded to other databases for verification. Increasing of the pattern numbers adds significant complexity for database scanning, even if we ignore the FP tree pruning cost. Taking the result in Table 6.2 as an example, when $\alpha=0.5\%$, the number of patterns generated from WS_1 is 49,660 are all needed for forwarding and verification by WS_2 (with over 1700 s scanning cost[3]). In the same setting, the dense database DS_1 generate more than six million rules requiring verification by DS_2 (this analysis explains why SQLP runs forever on DS databases for $\alpha\leq 0.5\%$). For PALP, all sites forward their patterns to the master site for verification,

[3] Database scanning is an expensive procedure. Our current implementation (using hash functions) can check about 30 patterns' frequencies over a 500 k transaction database in one second (the actual performance varies depending on pattern and transaction lengths).



	0.2	0.3	0.4	0.5	0.6	0.75	1
SQLP		74786.67	6327.34	2209.7	180.55	105.39	20.16
PALP		34798.51	1746.37	531.83	182.34	106.77	20.98
CLAP	1957.38	1014.23	648.35	411.78	184.06	108.56	21.94

Fig. 8. Query runtime comparison on Q_4 in Table 4 (WS databases).

creating a huge burden for the master site to compare and verify the patterns. In our implementation, the master site builds a bloom filter for patterns discovered from each site, so PALP avoids clause-level rule comparison and saves a tremendous amount of runtime, but it is still time consuming when the number of patterns is large.

Because of these reasons, the performance of both SQLP and PALP are inefficient when the support value α is 0.5% or smaller.

CLAP, although it is subject to overheads for pattern switching between sites, provides significantly better overall performance of the joint mining framework than self-contained mining frameworks. For relatively small α values (*e.g.*, $0.5\% \le \alpha \le 1.0\%$), CLAP linearly responds to the support value when answering the query.

CLAP's system runtime mainly consists of two parts: (1) the FP tree and bloom filter construction for each local site; and (2) CLAP cross-database pruning and pattern growth. As shown in Table 6.3, by switching bloom filters across sites, CLAP receives very significant performance gains for both SD and WS databases. Intuitively, CLAP is superior to PALP because it does not need to build a centralized data structure for cross-database pruning. In addition, because the mining activities of the distributed sites are collaboratively carried out in parallel, CLAP is superior to PALP on SD databases. Altogether, CLAP provides the best performance for both SD and WS databases.

Table 7 reports the system runtime for answering the first three queries of Table 4, confirming that CLAP provides the best performance for mining all three types (*L*-, *G*-, and *I*-) of patterns. One interesting finding is that CLAP is not only effective for *I*-pattern discovery, but is also effective for L-pattern mining (*e.g.* Q_1). This is because the bloom filter (which contains length-l patterns) built for each local site can be re-used during the local pattern growing process. For example, assume we have a bloom filter BF_3 for the local site S_1 and a base set BS = {abd}. When growing a pattern ϑ = BS∪g = {abdg} by using DLFP-Growth in Fig. 3, we can query and check whether a length-3 sub-set of ϑ, say {bdg}, exists in the BF_3 or not. According to the Apriori rule, we can stop growing ϑ if {bdg} does not exist in BF_3 and, therefore, speed up the pruning process. Traditional FP-Growth, however, does not have all length-3 patterns (due to its recursive pruning nature), and has to continuously grow ϑ. In our experiment, when $\alpha = 0.7\%$ the tree pruning time for FP_1 is 694.15 s (SD_1 database), whereas, by using a local BF_3 bloom filter, CLAP's pruning time is 271.46 s (in addition to 6 s for length-3 pattern discovery and BF_3 bloom filter construction), which is about a 40% runtime reduction!

Table 8 summarizes the overall performance of three frameworks for different databases and threshold values. The simple summary concludes that CLAP is suitable for any types of data and parameter settings. PALP is mostly effective if the support values are large, but deteriorates significantly for small support values (due to its self-contained mining nature). SQLP is the least attractive choice for mining distributed databases,

7. Conclusions

In this paper, we advocated that the essential goal for distributed pattern mining, from an association rule mining perspective, is to discover local, global, and inter patterns (namely *L*-, *G*-, and *I*-patterns). We argued that existing research mainly focuses on *L*- and *G*-pattern discovery, and has left *I*-pattern mining inadequately addressed, where single database oriented pattern pruning is essentially ineffective. More importantly, no existing framework is able to support the mining of all three types of patterns. We therefore proposed a distributed mining framework, namely collaborative pattern mining (CLAP), which is fully distributed with capability for cross-database pruning. The CLAP distributed mining framework has very little privacy concerns and requires low computational costs and memory consumption. Experimental comparisons demonstrated that CLAP significantly outperforms other simple methods.

The problem addressed in this paper mainly focuses on frequent item-set mining. However, the distributed mining framework and the cross-database pruning principles can be extended to handle other patterns, such as constrained frequent item-sets, closed frequent patterns, and sequential patterns.

Table 6.1
System runtime decomposition for SQLP, PALP, and CLAP to answer query $Q_4 = \{S_1 \ge S_2 \ge S_3 \ge S_4 \ge 0.5\%\}$ in Table 4.
Runtime decomposition for SQLP. The system runtime mainly consists of: (1) FP tree mining at the seed site S_1; and (2) database scanning for S_2, S_3, and S_4.

Databases		S_1	S_2	S_3	S_4	System runtime
WS	Seconds	431.43	1758.32	8.79	5.36	2209.7
	# Rules	49,660	378	37	36	
SD	Seconds	3528.3	185687[§]	19542.2	1031.1	209788.8
	# Rules	5582 k	1097 k	106 k	32881	

[§]Time estimated based on the average pattern search speed.

Table 6.2
System runtime decomposition for SQLP, PALP, and CLAP to answer query $Q_4 = \{S_1 \ge S_2 \ge S_3 \ge S_4 \ge 0.5\%\}$ in Table 4.
Runtime decomposition for PALP. The system runtime mainly consists of: (1) the maximum FP tree mining from S_1, S_2, S_3, and S_4; and (2) pattern comparison at the master site.

Databases		S_1	S_2	S_3	S_4	Master	System runtime
WS	Seconds	431.43	194.81	190.35	39.92	100.40	531.83
	# Rules	49,660	75,651	225,415	17,819	368,545	
SD	Seconds	3528.3	1356.4	719.6	393.8	1744.0	5283.39
	# Rules	5582 k	6230 k	6137 k	4836 k	22785 k	

Table 6.3
System runtime decomposition for SQLP, PALP, and CLAP to answer query $Q_4 = \{S_1 \ge S_2 \ge S_3 \ge S_4 \ge 0.5\%\}$ in Table 4.
Runtime decomposition for CLAP. The system runtime mainly consists of: (1) constructing bloom filters containing length-l patterns for each site; and (2) the maximum collaborative mining time on a site. ($l = 3$).

Databases		BF_1_l	BF_2_l	BF_3_l	BF_4_l	CLAP(S_1)	System runtime
WS	Seconds	405.94	164.29	87.84	30.93	5.84	411.78
	# Rules	958	2308	2829	1831	141	
SD	Seconds	51.08	26.21	14.28	7.95	448.43	499.51
	# Rules	30181	29493	30147	30301	40721	

Table 7
Query runtime comparison on Q_1, Q_2, and Q_3 in Table 4 ($\alpha = 0.5\%, \beta = 0.01\%$), a dash line indicates that a specific method is not capable of answering the query.

Frameworks	WS			SD		
	Q_1	Q_2	Q_3	Q_1	Q_2	Q_3
SQLP	859.52	–	–	6030.17	–	–
PALP	435.21	–	–	3531.30	–	–
CLAP	407.24	478.53	778.69	706.32	3604.02	4339.11

Table 8
The niche of the three mining frameworks. "+", "−", and "~" denotes that the framework in a specific row is effective, ineffective, or partially effective for conditions listed in the corresponding column.

Frameworks	Strong dense databases		Weak sparse databases	
	Small α	Large α	Small α	Large α
SQLP	−	~	−	~
PALP	−	+	−	+
CLAP	+	+	+	+

Acknowledgments

This research is supported in part by Australian Research Council (ARC) Future Fellowship under grant No. FT100100971, ARC Discovery Project under grant No. DP1093762, National Science Foundation of China Innovative Grant (70921061), and by the CAS/SAFEA International Partnership Program for Creative Research Teams.

References

[1] R. Agrawal, J.C. Shafer, Parallel mining of association rules, IEEE Transactions on Knowledge and Data Engineering 8 (6) (December 1996) 962–969.
[2] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, Proc. of VLDB Conference, 1994.
[3] M. Aounallah, G. Mineau, Distributed data mining: why do more than aggregating models, Proc. of IJCAI Conference, 2007, pp. 2645–2650.
[4] M. Ashrafi, D. Taniar, K. Smith, ODAM: an optimized distributed association rule mining algorithm, IEEE Distributed Systems Online 5 (3) (2004).
[5] S. Bay, M. Pazzani, Detecting group differences: mining Contrast sets, Data Mining and Knowledge Discovery 5 (3) (2001) 213–246.
[6] S. Bhattacharyya, S. Jha, K. Tharakunnel, J. Westland, Data mining for credit card fraud: a comparative study, Decision Support Systems 59 (3) (2011) 602–613.
[7] A. Border, M. Mitzenmacher, Network applications of bloom filters: a survey, Proc. of the 40th Annual Allerton Conf. on Communication, Control, and Computing, Urbana-Champaign, Illinois, 2002, pp. 636–646.
[8] C. Bucila, J. Gehrke, D. Kifer, W. Whote, DualMiner: a dual-pruning algorithm for itemsets with constraint, Proc. of ACM SIGKDD Conference, 2002.
[9] G. Buehrer, S. Parthasarathy, S. Tatikonda, T. Kurc, J. Saltz, Toward terabyte pattern mining: an architecture-conscious solution, Proc. of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming, 2007.
[10] B. Chazelle, J. Kilian, R. Rubinfeld, A. Tal, The Bloomier filter: an efficient data structure for static support lookup tables, Proc. of the 5th ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 30–39.
[11] B. Chen, L. Chen, Y. Lin, R. Ramakrishnan, Prediction cubes, Proc. of the 31st VLDB Conference, Norway, 2005.
[12] D. Cheung, V. Ng, A. Fu, Y. Fu, Efficient mining of association rules in distributed databases, IEEE Trans. on Knowledge and Data Engineering 8 (1996).
[13] S. Cohen, Y. Matias, Spectral bloom filters, Proc. of SIGMOD Conference, 2003, pp. 241–252.
[14] S. Datta, C. Giannella, H. Kargupta, K-means clustering over a large, dynamic network, Proc. of 2006 SIAM Conference on Data Mining, April 2006.
[15] G. Dong, J. Li, Efficient mining of emerging patterns: discovering trends and differences, Proc. of the 5th ACM SIGKDD Conference, 1999.
[16] L. Fan, P. Cao, J. Almeida, A. Broder, Summary cache: a scalable wide-area web cache sharing protocol, IEEE/ACM Trans. on Networking 8 (3) (2000) 281–293.
[17] W. Fujibuchi, T. Kato, Classification of heterogeneous microarray data by maximum entropy kernel, BMC:Bioinformatics (267) (2007) 8.
[18] A. Gionis, H. Mannila, P. Tsaparas, Clustering aggregation, Proc. of the 21st ICDE Conference, 2005.
[19] A. D'Costa, V. Ramachandran, A. Sayeed, Distributed classification of Gaussian space-time sources in wireless sensor networks, IEEE Journal on Selected Areas in Communications 22 (6) (2004) 1026–1036.
[20] X. Gong, W. Qian, Y. Yan, A. Zhou, Bloom filter-based XML packets filtering for millions of path queries, Proc. of ICDE Conference, 2005, pp. 890–901.
[21] J. Gray, A. Bosworth, A. Layman, H. Pirahesh, Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total, Proc. of the 12th ICDE Conference, 1996, pp. 152–159.
[22] R. Grossman, A top-ten list for data mining, SIAM News 34 (5) (2001).
[23] E. Han, G. Karypis, V. Kumar, Scalable parallel data mining for association rules, Proc. of ACM SIGMOD Conference, 1997.
[24] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidates generation, Proc. of ACM SIGMOD Conf., 2000.
[25] [25] IBM Quest Data Mining Project. Quest synthetic data generation code, http://www.cs.loyola.edu/~cgiannel/assoc_gen.html.
[26] [26] IPUMS: Integrated Public Use Microdata Series, http://www.ipums.umn.edu/usa/index.html.
[27] X. Ji, J. Bailey, G. Dong, Mining minimal distinguishing subsequence patterns with gap constraints, Proc. of ICDM Conference, 2005.
[28] R. Jin, G. Agrawal, A systematic approach for optimizing complex mining tasks on multiple databases, Proc. of ICDE Conference, 2006.
[29] M. Kantarcioglu, C. Clifton, Privacy-preserving distributed mining of association rules on horizontally partitioned data, Proc. of ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02), June 2002.
[30] [31] H. Kargupta et al., Distributed association rule mining bibliography, http://www.cs.umbc.edu/~hillol/DDMBIB/ddmbib_html/DistAss.html.
[31] H. Kargupta, B.H. Park, D. Hershberger, E. Johnson, Collective data mining: a new perspective toward distributed data mining, Advances in Distributed and Parallel Knowledge Discovery, MIT/AAAI Press, Cambridge, MA, 1999.
[32] T. Li, M. Ogihara, S. Zhu, Association-based similarity testing and its applications, Intelligent Data Analysis 7 (3) (2003) 209–232.
[33] S. Li, T. Wu, W. Pottenger, Distributed higher order association rule mining using information extracted from textual data, ACM SIGKDD Explorations 7 (1) (2005).
[34] P. Luo, H. Xiong, K. Lü, Z. Shi, Distributed classification in peer-to-peer networks, Proc. Of ACM KDD, 2007, pp. 968–976.
[35] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, C. Olston, Finding (recently) frequent items in distributed data streams, Proc. of ICDE Conference, 2005.
[36] S. Merugu, J. Ghosh, A distributed learning framework for heterogeneous data sources, Proc. of the 11th ACM KDD Conference, 2005.
[37] M. Otey, A. Veloso, C. Wang, S. Parthasarathy, W. Meira, Mining frequent itemsets in distributed and dynamic databases, Proc. of ICDM Conference, 2003.
[38] B. Park, H. Kargupta, Distributed data mining: algorithms, systems, and applications, in: Y. Nong (Ed.), Data Mining Handbook, 2002.
[39] S. Parthasarathy, M. Ogihara, Exploiting dataset similarity for distributed mining, Proc. of High Performance Data Mining Workshop, 2000.
[40] A. Partow, Open Bloom FilterSource code download:, http://bloom.googlecode.com/svn-history/r5/trunk/bloom_filter.h 2000.
[41] [41] K. Peeters, Tree.hh: an STL-like C++ tree class, http://www.aei.mpg.de/~peekas/tree/ September 2009.
[42] [42] F. Provost, Distributed data mining: scaling up and beyond. In Kargupta, H., Chan, P., eds.: Advances in Distributed and Parallel Knowledge Discovery, MIT/AAAI Press, 2000.
[43] L. Qiu, Y. Li, X. Wu, Preserving privacy in association rule mining with bloom filters, Journal of Intelligent Information Systems 29 (3) (2007) 253–278.
[44] A. Schuster, R. Wolff, Communication-efficient distributed mining of association rules, Data Mining and Knowledge Discovery 8 (2) (2004) 171–196.
[45] S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, D. Fan, P. Chan, JAM: Java agents for meta-learning over distributed databases, Proc. of KDD Conf., 1997, pp. 74–81.
[46] J. Sun, S. Papadimitriou, C. Faloutsos, Distributed pattern discovery in multiple streams, Proc. of PAKDD Conference, 2006, pp. 713–718.
[47] D. Tsur, J.D. Ullman, S. Abitboul, C. Clifton, R. Motwani, S. Nestorov, Query flocks: a generalization of association-rule mining, Proc. of ACM-SIGMOD Conference, 1998.
[48] J. Wang, H. Zeng, Z. Chen, H. Lu, L. Tao, W. Ma, ReCoM: reinforcement clustering of multi-type interrelated data objects, Proc. of SIGIR Conference, 2003, pp. 274–281.
[49] G. Webb, S. Butler, D. Newlands, On detecting differences between groups, Proc. of the 9th ACM SIGKDD Conference, 2003.
[50] X. Wu, S. Zhang, Synthesizing high-frequency rules from different data sources, IEEE Transactions on Knowledge and Data Engineering 15 (2) (2003) 353–367.
[51] K. Xu, S. Liao, J. Li, Y. Song, Mining comparative opinions from customer reviews for competitive intelligence, Decision Support Systems 50 (4) (2011) 743–754.
[52] Y. Yang, X.D. Wu, X. Zhu, Conceptual equivalence for contrast mining in classification learning, Data and Knowledge Engineering 67 (3) (2008) 413–429.
[53] X. Yin, J. Han, P. Yu, Crossminer: efficient classification across multiple database relations, Proc. of ICDE Conference, 2004.
[54] M. Zaki, Parallel and distributed association mining: a survey, IEEE Concurrency 7 (4) (1999).
[55] S. Zhang, M. Zaki, Mining multiple data sources: local pattern analysis, Data Mining and Knowledge Discovery 12 (2–3) (2006) 121–125.
[56] T. Zhang, R. Ramakrishnan, M. Linvy, BIRCH: an efficient data clustering method for very large databases, Proc. of ACM SIGMOD Conference, 1996.
[57] S. Zhang, C. Zhang, X. Wu, Knowledge discovery in multiple database, Springer, 2004.
[58] P. Zhang, X. Zhu, Y. Shi, L. Guo, X. Wu, Robust ensemble learning for mining noisy data streams, Decision Support Systems 50 (2) (2011) 469–479.
[59] X. Zhu, R. Jin, Multiple information source cooperative learning, Proc. of 21st International Joint Conference on Artificial Intelligence, 2009, pp. 1369–1376.

[60] X. Zhu, X. Wu, Discovering relational patterns across multiple databases, Proc. of ICDE Conference, 2007.
[61] X. Zhu, R. Jin, Y. Breitbart, G. Agrawal, MMIS-07, 08: mining multiple information sources workshop report, ACM SIGKDD Explorations 10 (2) (2008) 61–65.

Xingquan Zhu received his Ph.D degree in Computer Science from Fudan University, Shanghai China, in 2001. He is a recipient of the Australia ARC Future Fellowship and a Professor of the Centre for Quantum Computation & Intelligent Systems, Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS), Australia. Before joining the UTS, he was a tenure track Assistant Professor in the Department of Computer Science & Engineering, Florida Atlantic University, Boca Raton FL, USA (2006–2009), a Research Assistant Professor in the Department of Computer Science, University of Vermont, Burlington VT, USA (2002–2006), and a Postdoctoral Associate in the Department of Computer Science, Purdue University, West Lafayette IN, USA (2001–2002). Dr. Zhu's research mainly focuses on data mining, machine learning, and multimedia systems. Since 2000, he has published more than 110 referred journal and conference proceedings papers in these areas. Dr. Zhu is an Associate Editor of the IEEE Transactions on Knowledge and Data Engineering (2009-), a Program Committee Co-Chair for the 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2011), and a Program Committee Co-Chair for the 9th International Conference on Machine Learning and Applications (ICMLA 2010).

Bin Li received his PhD degree in Computer Science from Fudan University, Shanghai China, in 2009. He is a Postdoctoral Research Fellow at the Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS), Australia. Before joining the UTS, he worked as a research fellow at the Institut TELECOM SudParis, France. Dr Bin Li's research interests include Machine Learning and Data Mining as well as their applications to Web and Knowledge-based Information Systems and Social Media Mining.

Xindong Wu is a Professor of Computer Science at the University of Vermont (USA), and a Fellow of the IEEE. He holds a PhD in Artificial Intelligence from the University of Edinburgh, Britain. His research interests include data mining, knowledge-based systems, and Web information exploration. He has published over 200 refereed papers as well as 25 books and conference proceedings in these areas. His research has been supported by the U.S. National Science Foundation (NSF), the U.S. Department of Defense (DOD), the National Natural Science Foundation of China (NSFC), and the Chinese Academy of Sciences, as well as industrial companies including Microsoft Research, U.S. West Advanced Technologies and Empact Solutions.
Dr. Wu is the founder and current Steering Committee Chair of the IEEE International Conference on Data Mining (ICDM), the founder and current Editor-in-Chief of Knowledge and Information Systems (KAIS, by Springer), the Founding Chair (2002-2006) of the IEEE Computer Society Technical Committee on Intelligent Informatics (TCII), and a Series Editor of the Springer Book Series on Advanced Information and Knowledge Processing (AI&KP). He was the Editor-in-Chief of the IEEE Transactions on Knowledge and Data Engineering. He served as Program Committee Chair/Co-Chair for the 2003 IEEE International Conference on Data Mining, the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, and the 19th ACM Conference on Information and Knowledge Management.

Dan He is a Ph.D student in the Computer Science Department, University of California, Los Angles. He received his Master's degree from the University of Vermont in 2005. His research mainly focuses on pattern mining from sequence databases and Bioinformatics.

Chengqi Zhang received the PhD degree from Queensland University in 1991, followed by a Doctor of Science (DSc-Higher Doctorate) from Deakin University in 2002. He has been a research professor in information technology at The University of Technology, Sydney (UTS) since December 2001. He is currently the director of the UTS Research Centre for Quantum Computation and Intelligent Systems. In addition, he is the leader of the data mining program at the Australian Capital Market Cooperative Research Centre. Dr. Zhang's research interests mainly focus on data mining and its applications, especially domain driven data mining, negative association rule mining, and multi-database mining. He has published more than 200 research papers, including several in first-class international journals, such as Artificial Intelligence and IEEE and ACM Transactions. He has delivered 12 keynote/invited speeches at international conferences over the last six years. He has been chairman of the Australian Computer Society National Committee for Artificial Intelligence since November 2005. He is a fellow of the Australian Computer Society (ACS) and a senior member of the IEEE Computer Society. His personal web page can be found at: http://www-staff.it.uts.edu.au/~chengqi/.