J Inf Process Syst, Vol.14, No.3, pp.709~726, June 2018
https://doi.org/10.3745/JIPS.04.0075

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Contribution to Improve Database Classification Algorithms for Multi-Database Mining

Salim Miloudi*, Sid Ahmed Rahal*, and Salim Khiat*

Abstract

Database classification is an important preprocessing step for the multi-database mining (MDM). In fact, when a multi-branch company needs to explore its distributed data for decision making, it is imperative to classify these multiple databases into similar clusters before analyzing the data. To search for the best classification of a set of n databases, existing algorithms generate from 1 to $(n^2-n)/2$ candidate classifications. Although each candidate classification is included in the next one (i.e., clusters in the current classification are subsets of clusters in the next classification), existing algorithms generate each classification independently, that is, without taking into account the use of clusters from the previous classification. Consequently, existing algorithms are time consuming, especially when the number of candidate classifications increases. To overcome the latter problem, we propose in this paper an efficient approach that represents the problem of classifying the multiple databases as a problem of identifying the connected components of an undirected weighted graph. Theoretical analysis and experiments on public databases confirm the efficiency of our algorithm against existing works and that it overcomes the problem of increase in the execution time.

Keywords

Connected Components, Database Classification, Graph-Based Algorithm, Multi-Database Mining

1. Introduction

Many large organizations have different types of business distributed through their branches and each branch may have its own database that collects transactions continuously from customers. For business decision making, traditional data mining techniques called *mono-database mining* integrate all the data from these databases to amass a huge dataset for knowledge discovery. However, this leads to an expensive search cost for centralized processing and might disguise some important patterns (i.e., frequent itemset and association rules). To overcome the latter problems, these multiple databases have to be classified into clusters of similar databases according to their data correlation.

For example, if an inter-sate company has 10 branches including 5 branches for household appliances, 3 branches for clothing and 2 branches for food, we cannot apply existing data mining techniques over the union of the 10 branch databases. We might need to classify them into 3 clusters according to their type of business, and then we can analyze each database cluster individually.

Manuscript received November 20, 2015; accepted August 31, 2016.
Corresponding Author: Salim Miloudi (salim.miloudi@univ-usto.dz)
* Dept. of Computer Science, Faculty of Computer Science and Mathematics, University of Sciences and Technology-Mohamed Boudiaf (USTOMB) Oran, Algeria (salim.miloudi@univ-usto.dz, rahalsa2001@yahoo.fr, salim.khiat@univ-usto.dz)

Consequently the multi-database mining (MDM) [1-4] task becomes more manageable.

In order to classify transactional multiple databases, few algorithms have been proposed in the literature [5-8]. The most efficient algorithm in terms of accuracy and time complexity is due to Adhikari and Rao [7], which finds the best classification of a set of n database in $O(m \times n^2)$ time such that m $(1 \leq m \leq (n^2\text{-}n)/2)$ is the number of all candidate classifications. When $m = (n^2 - n)/2$ and n is very large, the time complexity will considerably increase. Hence, to reduce the time complexity of the existing algorithms, we present in this paper an improved algorithm that classifies a set of n databases $D=\{D_1,D_2,\ldots,D_n\}$ by analyzing and determining the connected components of an undirected weighted graph $G=(D,E)$, such that D is the vertex set and E the edge set (definitions are given in Section 3.1).

The rest of the paper is organized as follows: Section 2 discusses the existing works and point out their limitations. In Section 3, we describe the proposed approach for classifying the multiple databases. In Section 4, we perform some experiments to analyze and compare the proposed algorithm with the existing works in terms of accuracy and running times. Finally, Section 5 concludes this paper and highlights the future works.

2. Related Works

Traditional MDM process aims to integrate all the data from the multiple databases for knowledge discovery. This approach may not be a good strategy because of the large amount of irrelevant data involved during the process. To overcome the issue related to data irrelevancy, Liu et al. [9,10] have proposed a preprocessing technique called *database selection* to identify which databases are more likely relevant to a user query (Q). A relevance measure named *RF* is thus designed to identify relevant databases containing reliable information relative to the user application.

Database selection is a good strategy to reduce the amount of explored data while improving the quality of the patterns mined from the multiple databases. However, this approach depends on the user-application and when the multiple databases are mined without specifying any application, database selection cannot be applied.

The previous work motivated Wu et al. [5] to propose a new approach *independent-application* for classifying multiple databases. Thus, a similarity measure *sim* has been designed to group similar databases within the same cluster. The similarity between two databases D_i and D_j, denoted $sim(D_i,D_j)$, is calculated based on items shared between databases D_i and D_j. Such similarity measure could be useful to estimate the correlation between large databases, since extracting more information such as frequent itemsets could be costly in terms of CPU time. However, using items may produce low accuracy in finding the correct similarity between two databases. In fact, two databases are similar if they share many transactions.

Based on the previous similarity measure, an algorithm named *BestClassification* [5] has been proposed to search for the best classification of a set of n databases. The previous algorithm calls a procedure *GreedyClass* to generate a classification for each similarity threshold δ defined initially by the user. Hence, two databases D_i and D_j are similar if the similarity value $sim(D_i,D_j)$ is above δ. The time complexity of the proposed algorithm is $O(h \times n^4)$, such that n is the database number and h is the number of classification generated before obtaining the best classification. Even if the experiment

results shown in [5] prove that correct classifications are obtained for certain similarity thresholds, the time complexity of the algorithm remains high and becomes sever when the database number increases. Moreover, *BestClassification* fails to find the best classification when choosing an incorrect similarity step size as shown in [6]. In certain cases, the while-loop of the algorithm doesn't terminate and may generate an infinite loop without finding any classification. The latter problem is due to the strong dependence of the algorithm on the similarity step size, which is a user-input.

Inspired by the same concepts presented in [5], Li et al. [6] have modified the algorithm *BestClassification* [5] in order to optimize its time complexity and obtain the correct best classification of the multiple databases. In order to avoid missing the best classification (i.e., in case in which an incorrect step size has been selected), the distinct similarity values between the n databases are used as similarity thresholds to generate classifications. Thus, for each distinct similarity value sorted in the increasing order, a classification is produced. The time complexity of their improved algorithm is $O(h\times n^3)$. Although the experiments results show that the proposed algorithm is efficient and produces always the best complete classification, the similarity measure still need to be improved since it still depends on common items shared between databases. Moreover, the time complexity of the algorithm needs to be optimized.

In 2007, Adhikari and Rao [7] have proposed an approach for multi-database clustering that uses a more accurate similarity measure based on the support of frequent itemsets shared between databases. The time complexity of the proposed algorithm is $O(m\times n^2)$, such that n is the database number and m ($1 \le m \le n^2$) is the number of all candidate classifications. The experiment results obtained in [7] show that the proposed algorithm is effective and finds correct and optimal classifications only at few similarity levels. However, its time complexity needs to be improved especially when $m = n^2$, which may occur frequently when the similarity values between databases are distinct.

In 2013, Liu et al. [8] have proposed an approach for completely classifying databases based on the same similarity measure proposed in [7]. The proposed algorithm is simple to implement but the clustering isn't effective when the database number increases. In fact, the time complexity of the algorithm is $O(t\times 2^n\times n)$ such that n is the database number and t the distinct similarity values between the n databases.

According to this study, existing multi-database classification algorithms [5-8] produce hierarchical classifications that verify the following property. Let $class(D, \delta_i)$ and $class(D, \delta_j)$ be two candidate classifications of $D=\{D_1,D_2,\ldots,D_n\}$ generated at two consecutive similarity level δ_i and δ_j .Then for each cluster g_x in $class(D,sim, \delta_i)$, there is certainly a cluster g_y in $class(D,sim, \delta_j)$, such that $g_x \subset g_y$.

Despite of the latter property, the previous algorithms produce each classification independently, that is, instead to use the earlier clusters (i.e., generated in the previous classifications) to build the clusters of the next classification, they generate each classification starting from the initial state where each database forms one cluster $\{D_1\},\{D_2\},\ldots,\{D_n\}$. Consequently the existing algorithms are time consuming and do unnecessary work.

The above observations motivated us to design an improved database classification algorithm that overcomes the later problems. To achieve our goal, we need to generate each classification from the earlier classification generated so far. The proposed approach and algorithms are presented in the following section.

3. Proposed Approach and Algorithms

In this section, we describe the proposed approach for classifying the multiple databases. We assume that each database has been mined using algorithms for frequent itemset discovery [11-13].

Let $D=\{D_1,D_2,\ldots,D_n\}$ be the set of n database objects to classify, then the problem of generating a classification $class(D,sim,\delta)$ at a similarity level δ, can be described in terms of determining the connected components of an undirected weighted graph $G=(D,E)$.

We consider the database set $D=\{D_1,D_2,\ldots,D_n\}$ as the vertex set of G and E the edge set. In the following, the terms database and vertex are used interchangeably. The weight of an edge $(D_i,D_j) \in E$ is the similarity value between the corresponding databases $sim(D_i,D_j)$. At a certain similarity level δ, there is an edge connecting two vertices D_i and D_j if and only if, $sim(D_i,D_j) \geq \delta$.

Initially, the graph $G=(D,E)$ is empty with n disconnected vertices, i.e, $E=\emptyset$. Then, similarities are calculated between each vertex pairs (D_i,D_j) using an appropriate similarity measure, for $i,j=1$ to n. After that, edges are added to E starting with the vertex pairs having the highest similarity as follows.

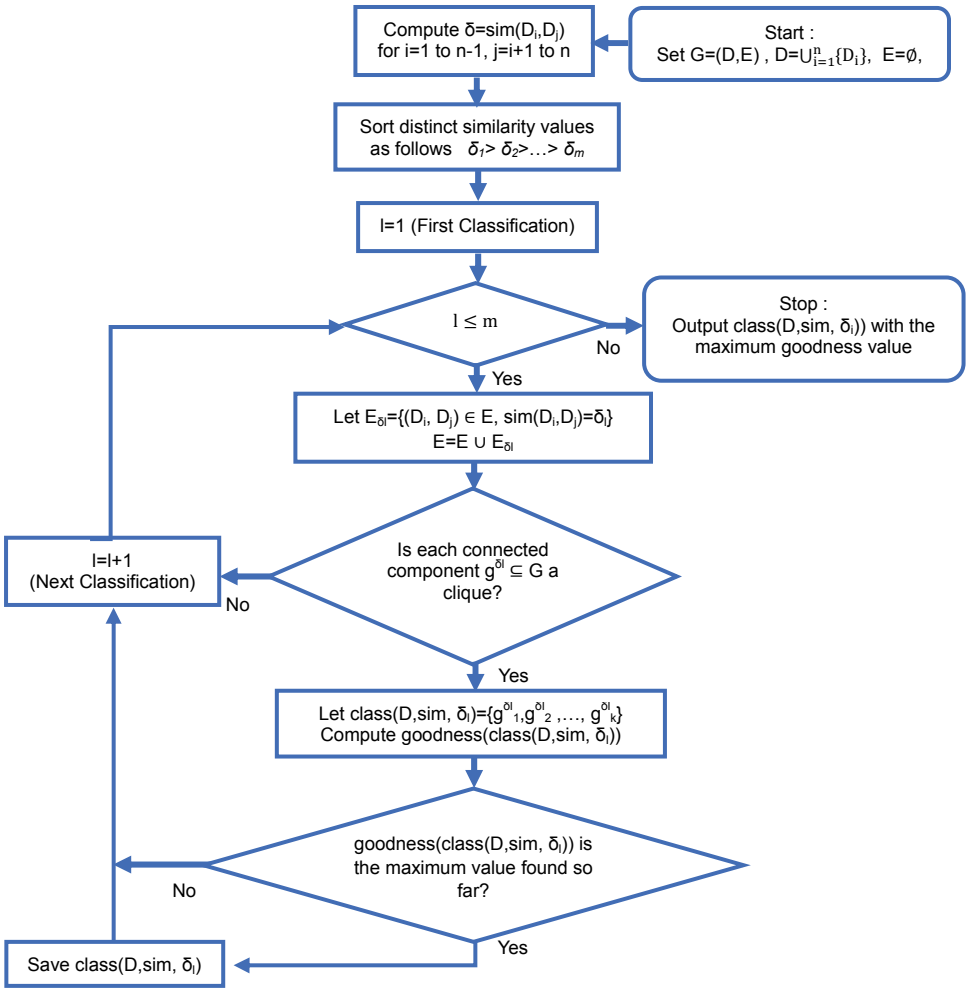


Fig. 1. Proposed approach for classifying the n multiple databases.

Let $\delta_1,\delta_2,\ldots,\delta_m$ be the m distinct similarity values between the n databases sorted in the decreasing order (i.e., $\delta_1>\delta_2>\ldots>\delta_m$) and let $E_{\delta l}=\{(D_i, D_j) \in E,\ sim(D_i,D_j)= \delta_l, i,j=1 \text{ to } n, i\neq j\}$ be the list of edges with weight value equal to δ_l (l=1 to m).

For each distinct similarity value δ_l, a candidate classification $class(D,sim,\delta_l)$ is generated by adding the edge list $E_{\delta l}$ to the edge set E such that $E=E_{\delta 1} \cup E_{\delta 2} \cup \ldots \cup E_{\delta l-1}$ (i.e., there are l–1 classifications generated earlier). Then, it remains to identify the connected component of G in order to discover the database clusters in $class(D,sim,\delta_l)$.

At a given similarity level δ_l, if each connected component of G, denoted $g_k^{\delta_l}$, forms a clique (i.e., a subset of vertices, each pair of which is connected by an edge in E), then the corresponding classification $class(D,sim, \delta_l)=\{ g_1^{\delta_l}, g_2^{\delta_l},\ldots, g_k^{\delta_l}\}$ is called a *complete* classification . In this paper we are interested in finding such classifications.

For classification assessment, the authors in [7] have proposed a measure *goodness* based on the intra-cluster similarity, the inter-cluster distance and the number of clusters. The complete classification with the maximum goodness value is selected as the best classification of the database set. Our classification approach is depicted in Fig. 1.

In the following section we give some definitions required to understand the proposed approach.

3.1 The Relevant Concepts

Referring to the related concept proposed by Adhikari and Rao [7], we have definitions as follows.

Definition 1. The similarity measure *sim* between two databases D_i, and D_j is calculated as follows.

$$sim(D_i, D_j, \alpha) = \frac{\sum_{I\in(FIS(D_i,\alpha)\cap FIS(D_j,\alpha))} min\{ supp(I,D_i) ,supp(I,D_j) \}}{\sum_{I\in(FIS(D_i,\alpha)\cup FIS(D_j,\alpha))} max\{ supp(I,D_i) ,supp(I,D_j) \}} \quad (1)$$

where α is the minimum support threshold, $I \in (FIS(D_i, \alpha) \cap FIS(D_j, \alpha))$ denotes that I is a frequent itemset in both D_i and D_j and supp(I, D_i) is the support of I in database D_i .The similarity measure *sim* has the following properties.

Property 1. $sim(D_i, D_j, \alpha)$ satisfies that :
(1) $0 \le sim(D_i, D_j, \alpha) \le 1$; (2) $sim(D_i, D_j, \alpha) = sim(D_j, D_i, \alpha)$;(3) $sim(D_i, D_i, \alpha) = 1$ for $i,j=1\ldots n$

Definition 2. The distance measure *dist* between databases D_i and D_j is defined as follows.

$$dist(D_i, D_j, \alpha) = 1 - sim(D_i, D_j, \alpha) \quad (2)$$

The distance function *dist* has the following properties.

Property 2. $dist(D_i, D_j, \alpha)$ satisfies that :
(1) $0 \le dist(D_i,D_j, \alpha) \le 1$; (2) $dist(D_i,D_j, \alpha) = dist(D_j,D_i, \alpha)$; (3) $dist(D_i,D_i, \alpha) = 1$ for $i,j=1\ldots n$

Definition 3. Let $class(D,sim,\delta)=\{g^{\delta}_1, g^{\delta}_2,\ldots,g^{\delta}_k\}$ be a classification of k clusters of the database set $D=\{D_1,D_2,\ldots,D_n\}$ at the similarity level δ. The intra-cluster similarity of $class(D,sim,\delta)$ is defined as follows.

$$intra\text{-}sim(class(D,\delta)) = \sum_{l=1}^{k} \sum_{D_i, D_j \in g_l^{\delta}}^{i \neq j} sim(D_i, D_j, \alpha) \quad (3)$$

DEFINITION 4. The inter-cluster distance of $class(D,sim,\delta)$ at the similarity level δ is defined as follows.

$$inter\text{-}dist(class(D,\delta)) = \sum_{g_p^{\delta}, g_q^{\delta}}^{p \neq q} \sum_{D_i \in g_p^{\delta}, D_j \in g_q^{\delta}}^{i \neq j} dist(D_i, D_j, \alpha) \quad (4)$$

The best classification is selected based on maximizing both the intra-cluster similarity and the inter-cluster distance. Hence, we have the following definition of the goodness measure.

DEFINITION 5. The goodness of $class(D,sim,\delta)$ is defined as follows.

$$goodness(class(D,\delta)) = |intra\text{-}sim(class(D,\delta)) + inter\text{-}dist(class(D,\delta)) - k| \quad (5)$$

The classification that gets the maximum value of *goodness* is selected as the best classification at the similarity level δ.

DEFINITION 6. Let $G=(D,E)$ be the graph representing the database set D. The vertex set $D=\{D_1, D_2,\ldots,D_n\}$ is the set of nodes in the graph G and $E=D\times D=\{(D_i, D_j), 1\leq i,j\leq n\}$ is the set of pairs (D_i, D_j) representing the links between vertices in G.

DEFINITION 7. At a similarity level δ, a classification $class(D,sim,\delta)$ is complete if the following equation is verified :

$$\sum_{i=1}^{k} \frac{|g_i^{\delta}|^2 - |g_i^{\delta}|}{2} = |E| \quad (6)$$

Where $|g_i^{\delta}|$ denotes the number of vertices in the connected component g_i^{δ} and $|E|$ is the number of edges in the graph G. In other words, if each class g_i^{δ} (i=1 to k) in $class(D,sim,\delta)$ is a clique in G then $class(D,sim,\delta)$ is a complete classification.

The proposed approach will be followed and explained with the help of examples.

EXAMPLE. Let $D=\{D_1, D_2,\ldots, D_6\}$ be the database set to classify. For illustration purposes, the similarity values between the six databases are given in Table 1.

Table 1. The similarity relation table for the six databases of the example

sim	D_1	D_2	D_3	D_4	D_5	D_6
D_1	1	0.40	0.56	0.40	0.50	0.40
D_2	-	1	0.40	0.79	0.40	0.45
D_3	-	-	1	0.40	0.45	0.40
D_4	-	-	-	1	0.40	0.45
D_5	-	-	-	-	1	0.40
D_6	-	-	-	-	-	1

From the upper triangle of the similarity table, there are five distinct similarity values. Consequently, there are five lists of edges with the same weight. Starting with the highest similarity, the five similarity values are sorted in the decreasing order as follows (0.79, 0.56, 0.50, 0.45, 0.40).

Let $G=(D,E)$. Initially $|D|=6$ and $E=\emptyset$. Therefore, a trivial complete classification is obtained $class(D)=\{\{D_1\},\{D_2\},\{D_3\},\{D_4\},\{D_5\},\{D_6\}\}$, with $goodness(class(D))= 2.20$.

For each similarity value δ listed in the decreasing order, a candidate classification $class(D,sim,\delta)$ is generated and checked to see whether it is a complete classification as follows.

For $\delta=0.79$, we have, $E_{0.79}=\{(D_2, D_4)\}$, $E=E \cup E_{0.79}$

$class(D,sim,0.79)=\{\{D_2,D_4\},\{D_1\},\{D_3\},\{D_5\},\{D_6\}\}$, which is a complete classification because :

$$\sum_{i=1}^{k}\frac{|g_i^{0.79}|^2-|g_i^{0.79}|}{2}=\frac{|\{D2,D4\}|^2-|\{D2,D4\}|}{2}+\frac{|\{D1\}|^2-|\{D1\}|}{2}+\frac{|\{D3\}|^2-|\{D3\}|}{2}+\frac{|\{D5\}|^2-|\{D5\}|}{2}+\frac{|\{D6\}|^2-|\{D6\}|}{2}=1=|E|$$

with $goodness(class(D,0.79))= 3.78$

The remaining classifications are summarized in Table 2.

There is only one classification that isn't complete at $\delta=0.50$. This is because, $|E|=3$ and $\sum_{i=1}^{k}\frac{|g_i^{0.50}|^2-|g_i^{0.50}|}{2}=4\neq|E|$

According to the results in Table 2, the best complete classification of the 6 databases that has the maximum value of goodness is $class(D,sim,0.45)=\{\{D_2, D_4, D_6\},\{D_1, D_3, D_5\}\}$

In the following section, we describe the data structures and algorithms used in the proposed classification approach.

Table 2. Candidate classifications generated from the similarity matrix in Table 1

Similarity level (δ)	Edge set (E_δ)	Candidate classification $class(D,sim,\delta)$	goodness($class(D,\delta)$)
0.56	(D_1, D_3)	{D_2,D_4},{D_1,D_3},{D_5},{D_6}	4.90
0.50	(D_1, D_5)	{D_2,D_4},{D_1,D_3,D_5},{D_6}	5.90
0.45	(D_2,D_6),(D_3,D_5),(D_4, D_6)	{D_2,D_4,D_6},{D_1,D_3,D_5}	6.60
0.40	(D_1,D_2), (D_1,D_4), (D_1, D_6), (D_2,D_3), (D_2,D_5), (D_3, D_4), (D_3,D_6), (D_4,D_5), (D_5, D_6)	{D_1,D_2,D_3,D_4,D_5,D_6}	5.80

3.2 Data Structure and Algorithms

The problem of generating a classification $class(D,sim,\delta)$ at a given similarity level δ depends mainly on the following sub-problems:

1) Find the list of edges with weight value equal to δ, denoted E_δ, and add it to the graph G.
2) Determine the connected components of G and check whether each of which forms a clique.

To solve the first problem, we need to use a dictionary data structure to efficiently search for duplicate weights. In our approach, we need to list the m edge lists E_δ in the decreasing order of the weight δ. Therefore, we use a balanced binary search tree (BST) such as red-black trees [14], to implement such ordered dictionary structure.

In the following section, we present a procedure to build the m-node BST where each node represents one edge list E_δ. Instead to use pointers to link edges with the same weight value, we use integers to

reference the edges and link them into a contiguous table of size $(n^2-n)/2$.

3.2.1 Finding the edge lists E_δ

Let T be the balanced BST object and T.*root* the root node. Each node in T contains two fields: *weight* and *index*, which represent the weight value δ and the index of the first edge in E_δ respectively. Edges with the same weight value δ are linked into a contiguous table V[1..$(n^2-n)/2$]. For example, let x be a node object in T, then V[x.*index*] contains the index of the second edge whose weight value is x.*weight* and V[V[x.*index*]] contains the index of the third one and so on, until we find a negative value (-1) , which represents the end of the list E_δ.

In the following procedure, we present the algorithm used to construct the ordered edge lists E_δ.

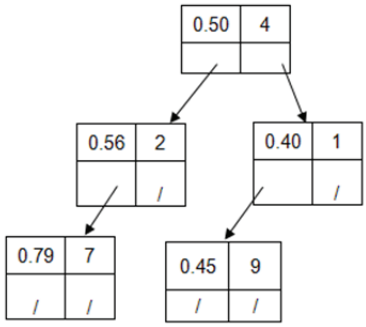
Procedure *Build _Edge_Tree*(*n,* M)
Input : n : the number of databases; M[nxn]: the similarity table of the database set D;
Output : T :the tree of edge lists;
V[1..$(n^2-n)/2$] : the index array used to link edges with the same weight;
dist_inter : the inter-cluster distance of $\{\{D_1\},\{D_2\},\ldots,\{D_n\}\}$
01. **let** *dist_inter*=0;
02. **let** T.*root*=null
03. **let** k=1;{// k is the index of the current edge//}
04. **for** i=1 **to** n-1, j=i+1 **to** n **do**
05. **let** δ =M[i,j] {//get the similarity between D_i and D_j//}
06. **let** *dist_inter* = *dist_inter* + (1– δ);
07. **let** V[k]= -1;
08. **let** x=***Tree_Search***(T.*root*, δ);
09. **if** $x\neq$ null and x.*weight*=δ **then**
10. **let** V[k]=x.*index;*
11 **let** x.*index*=k; {//Add the k-th edge to the list E_δ//}
12. **else**
13. Allocate a new node object y;
14. **let** y.*weight*=δ;
15. **let** y.*index*=k;
16. ***Tree_Insert***(T,y){//Insert the node y into the tree T //}
17. ***Tree_Rebalance***(T) {//Rearrange the tree after the insertion operation//}
18. **end if**
19. **let** k=k+1;
20. **end for**
21. return (T, V[0..C_n^2-1], *dist_inter*);
end procedure

Discussion. Initially (at line 2), the tree T is empty. Then for each databases pair (D_i,D_j) (at line 4), a binary search tree is performed at line 8 to see whether δ exists in T. If there is a node x in T such that x.*weight* = δ then the current edge (i.e., the k-th edge) is inserted at the beginning of E_δ at lines 10–11. Otherwise, a new node, say y, is allocated and inserted into T at lines 13–16 such that y.*weight* is set to δ and y.*ndex* is set to k. The auxiliary procedure *Tree_Rebalance* is called to re-balance the tree after the insertion operation in order to keep the height proportional to the logarithm of the number of nodes in T. The implementation of *Tree_Rebalance* depends on the balance information and the algorithm used

to re-balance the tree when certain properties are violated [14].

Let m be the number of nodes in T such that $m \in [1,(n^2-n)/2]$. Then, the search, insert and rebalance operations take $O(log_2(m))$. There are $(n^2-n)/2$ similarity values between the n databases. For each similarity value δ a new node is inserted into T if there is no node containing the weight δ. Hence, the time complexity to build T is $O(n^2 log_2(m))$.

The tree T and the index array V corresponding to the example above are shown in Fig. 2.



k	edge	edge_weight	index (next edge)
1	(D_1, D_2)	0.40	3
2	(D_1, D_3)	0.56	-1
3	(D_1, D_4)	0.40	5
4	(D_1, D_5)	0.50	-1
5	(D_1, D_6)	0.40	6
6	(D_2, D_3)	0.40	8
7	(D_2, D_4)	0.79	-1
8	(D_2, D_5)	0.40	10
9	(D_2, D_6)	0.45	11
10	(D_3, D_4)	0.40	12
11	(D_3, D_5)	0.45	14
12	(D_3, D_6)	0.40	13
13	(D_4, D_5)	0.40	15
14	(D_4, D_6)	0.45	-1
15	(D_5, D_6)	0.40	-1

Fig. 2. The balanced binary search tree T and the index array V (the column index) of the example.

Each node of T contains mainly 4 fields (the similarity value δ, the index of the first edge in the list E_δ, the left and right child pointers). The table to the right represents the array V[1..$(n^2-n)/2$] in which edges with the same weight are linked. Only the most right column (the column index) exists really in the main memory. Other columns are only shown for illustration purpose.

PROPOSITION. Let (D_i,D_j) be the vertex pair corresponding to the k-th edge for i=1 to n–1, j=i+1 to n and k=1 to $(n^2-n)/2$. Then, the row index i and the column index j of the k-th edge are calculated as follows:

$$i = n - h \tag{7}$$

$$j = (i+1) + h(h+1)/2 - (C_n^2 - k + 1) \tag{8}$$

$$\text{With } h = \left\lceil \left(\sqrt{8 \times (C_n^2 - k + 1) + 1} - 1\right)/2 \right\rceil$$

Such that $\lceil val \rceil$ is the least integer greater than or equal to val and $C_n^2 = (n^2-n)/2$.

Proof. Let M be the triangular similarity table of size $n \times n$. Then, there are C_n^2 similarity values in M, which are associated with C_n^2 edges indexed from 1 to C_n^2. The 1st row of M is associated with n–1 edges $(D_1,D_2),(D_1,D_3),\ldots(D_1,D_n)$, the 2nd row with n–2 edges $(D_2,D_3),(D_2,D_4),\ldots(D_2,D_n)$, and so on until the last row, the $(n-1)$-th row, which is associated with one edge (D_{n-1},D_n). Thus, the last h rows of M are associated with $h(h+1)/2$ edges. For a given index k, there are C_n^2-k+1 edges counting from the k-th edge

to the last edge, the C_n^2-th edge. To find the row index i of the k-th edge, we have to subtract h from n, such that h is the integer value that verifies: $h(h+1)/2 \geq C_n^2 - k + 1$. By solving the inequality we find: $(h+1/2)^2 \geq (8(C_n^2 - k + 1) + 1)/4$, $h \geq \left(\sqrt{8 \times (C_n^2 - k + 1) + 1} - 1\right)/2$. Therefore, $i = n - h$ such that $h = \left\lceil (\sqrt{8 \times (C_n^2 - k + 1) + 1} - 1)/2 \right\rceil$. The first column in the row i has the index i+1. To find the column index j of the k-th edge, we have to add the offset $(h(h+1)/2 - (C_n^2 - k + 1))$ to the column index (i+1).

3.2.2 The database classification algorithm

To solve the second problem, which is determining and maintaining the connected components of G, we use the *disjoint-set forest* data structure [14]. This data structure keeps a collection of n disjoint updatable trees, where each tree represents one connected component in the graph G=(D,E). The root node of each tree is used as a representative node to identify one connected component. Each node in the disjoint-set forest, say x_i (i=1 to n), contains two main fields *size* and *parent*, which represent respectively the size of the sub-tree rooted at x_i and a pointer to the parent node. In this paper, instead to use pointers, we use integers to link child nodes to their parent nodes. For example, if x_j is the parent of x_i, then we set x_i.*parent*=j. The data structure has two main operations described as follows.

union(x_i, x_j) : combines the trees identified by the roots x_i and x_j. That is, it links the root of the smaller tree to the root of the larger tree. After a *union* operation, the size of the resulting tree is set equal to the sum of the sizes of the input trees.

findset(x_i) : returns the root of x_i's tree, that is, it follows parent nodes starting from x_i's parent, until reaching the root node. For each edge incident on the i-th vertex, *findset* is called on the node x_i to find the connected component to which the i-th vertex belong.

In this paper, we have modified the *findset*'s implementation in order to identify cliques in G=(D,E). If x_i or x_i's parent (say x_j) is the root, then *findset*(x_i) returns the root node. Otherwise, x_i is linked to x_j's parent and *findset*(x_i) outputs x_j's parent even if it's not the root. Thus, for each edge incident on the i-the vertex, *findset*(x_i) compresses the path between x_i and the root when x_i and x_i's parent aren't the root. Therefore, if x_i's tree represents a connected component of k vertices in G and *findset*(x_i)'s output isn't the root, then it is clear that there is at least one missing edge between the i-th vertex and the remaining (k-1) vertices of the same component. Consequently, the connected component, which contains the i-th vertex, isn't a clique in G.

In the following we present our classification algorithm that takes as input the objects returned by the procedure *BuildEdgeTree*. The *findset*, *union* and *makeset* procedures are given too.

Algorithm *BestDatabaseClustering*(n,D,T,V, *dist_inter*)

Input : n : the number of databases; D={D_1,D_2,…,D_n} : the database set;
T : balanced binary search tree of all edge lists;
V[1..(n^2-n)/2] : the index array used to link edges with the same weight;
dist_inter : the inter-cluster distance of {{D_1},{D_2},…,{D_n}}
Output : *class*(D,*sim*,δ) : the best complete classification;
01. **let** A[1..n] be the array containing the disjoint components of G=(D,E);
02. **let** *nb_component* =n;
03. **let** *sim_intra*= 0; //the intra-cluster similarity

04. **let** *goodness_max*= |*dist_inter*+ *sim_intra* – *n*|;
05. **let** *nb_edges*=0;//number of edges examined so far
06. **let** *nb_clique_edges*=0;//number of edges required to produce cliques in *G*=(*D,E*);
07. **let** *best_class_version*=1; **let** *class_version*=1;
08. ***makeset***(A);
09. **for each** node *e* ∈T taken in the decreasing order by field *weight* **do**
10. **let** *k*= *e.index;*
11. **let** *δ*= *e.weight;*
12. **while** *k*≠ – 1 **do**
13. **let** (*D*$_i$,*D*$_j$) be the end-vertices corresponding to the *k*-th edge;
14. increase *nb_edges* by one;
15. increase *best_class_version* by one;
16. **let** *dist_inter* = *dist_inter* – (1– *δ*); **let** *sim_intra* = *sim_intra* + *δ*;
17. **let** xi=***findset***(A[*i*], *best_class_version*, *class_version*);
18. **let** xj=***findset***(A[*j*], *best_class_version*, *class_version*);
19. **if (**xi.*parent*= – 1 and xj.*parent*= – 1 and xi≠ xj) **then**
20. **let** *nb_clique_edges* = *nb_clique_edges* +(xi.*size*×xj.*size*);
21. ***union*(**xi , xj, *best_class_version*, *class_version*)**;**
22. *nb_component*= *nb_component* – 1;
23. **end if**
24. **let** *k*=V[*k*];
25. **end while**
26. **if** *nb_edges*≠ *nb_clique_edges* **then**
27. output **"***class*(*D,sim,δ*) is not a complete classification**"**
28. **else**
29. **let** *temp*=|*sim_intra* +*dist_inter* – *nb_component*|;
30. **if** *temp*>*goodness_max* **then**
31. **let** *goodness_max*=*temp*;
32. **let** *best_class_version*=*class_version*;
33. **end if**
34. **end if**
35. **end for**
36. **for** *i*=1 **to** *n* **do**
37. **if** A[*i*].*version*≤*best_class_version* **then**
38. **output "**A[*i*].*parent* is the cluster index of the *i*-th database"
39. **else output "**A[*i*].*parent_temp* is the cluster index of the *i*-th database"
40. **end if**
41. **end for**

end algorithm

Procedure *makset*(A**)**

Input : A[1..*n*]: the array representing the *n* nodes of the disjoint-set forest

01. **for** *i*=1 **to** *n* **do**
02. **let** A[*i*]= new node object xi; *//allocate a new node object*
03. **let** xi.*parent*= – 1;
04. **let** xi.*size*=1;
05. **let** xi.*parent_temp*= – 1;
06. **let** xi.*version*=1;
07. **end for;**

end procedure

Procedure *union*(x_i, x_j,*class_version,best_class_version***)**

Input : xi, xj: two disjoint components to merge
class_version : the version of the current classification
best_class_version : the version of the best classification

01. **if** xi.*size*>xj.*size* **then**
02. **if** xj.*version* ≤ *best_class_version* **then let** xj.*parent_temp*= xj.*parent;***end if**
03. **let** xj.*parent*=i;
04. **let** xj.*version*=*class_version,//*update the current parent version of xj
05. **let** xi.*size*=xi.*size*+xj.*size*;
06. **else**
07. **if** xi.*version*≤*best_class_version* **then let** xi.*parent_temp*=xi.*parent;* **end if**
08. **let** xi.*parent*=j;
09. **let** xi.*version*=*class_version,//*update the current parent version of xi
10. **let** xj.*size* =xj.*size* +xi.*size*;
11. **end if**
end procedure

Procedure *findset*(x_i, *class_version,best_class_version***)**

Input : A[1..*n*]: the array representing the *n* nodes of the disjoint-set forest
xi : a node object
class_version : the version of the current classification
best_class_version : the version of the best classification
Output :the root or the closest node from the root.
01. **if** xi.*parent* = – 1 **then** return xi;
02. **else if** A[xi.*parent*].*parent* = – 1 **then** return A[xi.*parent*];
03. **else**
04. **if** xi.*version*≤*best_class_version* **then** let xi.*parent_temp*=xi.*parent;* **end if;**
05. **let** xi.*parent*= A[xi.*parent*].*parent*;
06. **let** xi.*version*=*class_version,//*update the current parent version x_i.*version;*
07. return A[xi.*parent*];
08. **end if;**
end procedure

Discussion. Initially, the graph $G=(D,E)$ has n disjoints graph components. Therefore, we get a trivial classification $class(D,sim)=\{\{D_1\},\{D_2\},\ldots,\{D_n\}\}$. Lines 1-7 initialize the variables used in our algorithm. The inter-cluster distance (*dist_inter*) between the n components is calculated during the call of the procedure *BuildEdgeTree*. The intra-cluster similarity (*sim_intra*) has a zero value since there is one database in each cluster. We use the variable *goodness_max* to maintain the maximum value of *goodness* found so far.

At line 8, the procedure *makeset* initializes the data structure by creating n rooted trees x_i (i=1 to n), each of which contains one node ($x_i.size$=1). To access the n nodes, we use an array A[1..n] such that A[i] holds the node object x_i representing the i-th vertex in G. The time complexity of line 8 is $O(n)$.

At line 9, the for-loop examines each node e of the edge tree (T) taken in the decreasing order by field *weight*. There are m nodes in T such that $1 \leq m \leq (n^2-n)/2$ and each node represents one edge list E_δ. Browsing all the m nodes in the sorted order takes $O(m)$ using an in-order traversal of T.

At line 12, the while-loop examines the current edge list E_δ such that δ = e.*weight*. Edges are listed

starting from the first edge, whose index is e.*index*, until reaching a negative value (–1), which represents the end of the edge list. At line 13, we use Eqs. (7) and (8) to find the two end vertices (D_i,D_j) corresponding to the k-th edge (i.e., the current edge).

At lines 17–18, the *findset* procedure is called once for each end-vertex of the k-th edge (D_i, D_j) to find the connected components to which D_i and D_j belong. The time complexity of lines 17-18 is $O(1)$.

Let x_i and x_j be the nodes returned by *findset*. At line 19, if x_i and x_j are two distinct roots, then the *union* procedure (at line 21) will merge the corresponding trees by making the smaller tree a sub-tree of the larger tree. Otherwise, either the current classification is not complete or the two vertices D_i and D_j already belong to the same component.

At line 20, we update *nb_clique_edges*, which is the number of edges required so that each connected component of G=(D,E) becomes a clique. After each union operation, we decrement the number of components *nb_component*.

At line 24, we select the next edge in the list E_δ from the index array V[1..(n^2–n)/2].

At line 26, we test whether the current classification is complete. If the number of edges examined so far, *nb_edges*, is equal to *nb_clique_edges*, then the current classification is complete and the algorithm continues at the line 29. Otherwise, the current classification is discarded.

At lines 29–31, the current complete classification *class*(D,*sim*,δ) is assessed using the goodness measure proposed in [7]. While *goodness*(*class,* δ) $\geq$ *goodness_max*, *class*(D,*sim*,δ) is the best classification generated so far and *goodness max* value is updated.

During the course of our algorithm, we need to maintain the best classification generated so far as the disjoint-set forest is updated. To implement such persistent data structure, additionally fields (*version* and *parent_temp*) are associated with each node in the disjoint-set forest. These fields are required to maintain a past version of the dynamic set at the time that the best classification was generated. Let x be a node in the disjoint-set forest. The field x.*parent_temp* is used to keep track of x's parent node corresponding to the best classification generated so far. The field x.*version* is used to determine the classification to which x's parent refers.

In our algorithm, each classification *class*(D,*sim*, δ) is identified by a version number *class_version*. Let *best_class_version* be the version number of the best classification. Before each updating operation in the *union*/*findset* procedure, each node x is checked to see whether its current parent is a part of the best classification produced so far. That is, if x.*version*$\leq$*best_class_version*, then x's parent index is saved in the field x.*parent_temp* before the updating operation.

At lines 36–41, we display the best classification of the n databases from the disjoint-set forest. Hence, we output the parent index of each node x_i ($1 \leq i \leq n$) at the time that the best classification was generated. That is, if x_i.*version*$\leq$*best_class_version*, then x_i.*parent* is returned as the cluster containing the i-th database. Otherwise, x_i.*parent_temp* is returned instead.

The algorithm terminates once all the m edge lists E_δ are examined and the m candidate classifications are generated. The average size of each edge list is n^2/m. Therefore, exploring all the m edge lists takes $O(n^2)$ time such that n is the database number. Therefore, the proposed algorithm finds the best complete classification of the set of n databases in $O(n^2)$ time. Our algorithm is optimal when comparing with *BestDatabasePartition* [7], which takes $O(m \times n^2)$ time to find the best partition of the multiple databases.

The space complexity of our algorithm is estimated as follows. To store the m nodes of T, we need

m×(2pointers+real+integer) units. We have to add the space required to store the array index V, that is, (n^2–n)/2 integer units. Also, the disjoint-set forest data structure needs n×(4integer) units. Therefore, the memory space required by our algorithm is ($2n^2$+14n+28m) bytes when using a typical compiler that represents a pointer, an integer and a real number using 8 bytes, 4 bytes and 8 bytes respectively.

In parallel, *BestDatabasePartition* needs n^2 real units to store the similarity relation table. Assuming that *BestDatabasePartition* uses a balanced binary search tree to sort all the distinct similarity values, then m×(2pointers+real) units will be required to store the nodes of the tree. Each candidate classification is stored separately. Hence, mn×integer units are required to store all the m candidate classifications. Therefore, the memory space required by *BestDatabasePartition* is ($8n^2$+20m+4mn) bytes by using the same compiler. For a large values of n and m, we notice that the proposed algorithm consumes less memory space compared with *BestDatabasePartition*.

4. Experiments

To assess the performance of our proposed approach, we conducted some experiments to analyze and compare the execution time of the existing algorithm *BestDatabasePartition* [7] and our proposed graph-based classification algorithm. All the experiments have been implemented on a 2.70-GHz Pentium processor with a 2-GB of memory, using JAVA Edition 6.

We carried out the experiments using two synthetic datasets T10I4D100K and T40I10D100K available in: http://fimi.ua.ac.be/data/. Some characteristics of these datasets are presented in Table 3. We denote by NT, ALT, AFI, NI and DB the number of transactions, the average length of a transaction, the average frequency of an item, the number of items and the database name respectively.

For multi-database mining, each of the datasets is divided horizontally into 10 and 20 databases. The multiple databases obtained from T10I4D100K and T40I10D100K are referred to as $T_{1,1}$, $T_{1,2}$, …, $T_{1,n}$ and $T_{4,1}$, $T_{4,2}$, …, $T_{4,n}$ respectively, such that n=10, 20.

Table 3. Synthetic datasets

DB	NT	ALT	NI	AFI
T10I4D100K	100,000	11.102280	870	1276.124138
T40I10D100K	100,000	40.605070	942	4310.516985

By varying the value of the minimum support threshold (α), we get different sets of frequent itemsets using FP-Growth algorithm [13] as shown in Table 4. We denote by {FIS($T_{i,1}$, α), FIS($T_{i,2}$, α),…, FIS($T_{i,n}$, α)} the sets of frequent itemsets reported from the n multi-databases $T_{i,1}$, $T_{i,2}$, …, $T_{i,n}$ under α such that i=1,4.

Once the data-preparation is done, we classify the multi-databases using our algorithm and *BestDatabasePartition* [7]. Since it is difficult to measure the performance of an algorithm executed by the Java Virtual Machine, due to the optimization that occurs during the execution, each algorithm is executed 1000 times on the same multiple databases to get the average running time.

In the first part of our experiments, we analyzed the impact of minsupp(α) on the classification execution time. Therefore, we set the number of databases n=10 and we varied the value of α to get

different sets of frequents itemsets. Then, we executed our algorithm and *BestDatabasePartition* on the same frequents itemsets. The classification results and the execution times are presented in Table 5 and Fig. 3, respectively. From the results, we can notice that the execution time is relatively constant for both algorithms when the value of α increases. The reason is that n and the number of candidate classification (m=45) didn't change during the experiments, as shown in Table 5. Moreover, we didn't take into account the overhead of calculating the similarities between the n databases, since the same similarity measure proposed in [7] has been used for the two algorithms. However, we observe that the execution time of our algorithm is shorter than that of *BestDatabasePartition* for α=0.03 to 0.06.

Table 4. The sets of frequent itemsets from the multiple databases $T_{1,1}$, $T_{1,2}$, …, $T_{1,n}$ and $T_{4,1}$, $T_{4,2}$, …, $T_{4,n}$ for α=0.03 to 0.06 and n=10, 20

Dataset (i=1,4)	# of multiple databases (n)	# of transaction in each database	Algorithm	Minimum support (α)	Sets of frequent itemsets
T_i	10	10,000 (×10)	FP-Growth	0.03	FIS($T_{i,1}$, 0.03), FIS($T_{i,2}$, 0.03), …,FIS($T_{i,10}$, 0.03)
				0.04	FIS($T_{i,1}$, 0.04), FIS($T_{i,2}$, 0.04), …,FIS($T_{i,10}$, 0.04)
				0.05	FIS($T_{i,1}$, 0.05), FIS($T_{i,2}$, 0.05), …,FIS($T_{i,10}$, 0.05)
				0.06	FIS($T_{i,1}$, 0.06), FIS($T_{i,2}$, 0.06), …,FIS($T_{i,10}$, 0.06)
	20	4,500 (×20)		0.03	FIS($T_{i,1}$, 0.03), FIS($T_{i,2}$, 0.03), …, FIS($T_{i,10}$, 0.03),…, FIS($T_{i,20}$, 0.03)

Table 5. A comparison between the classification results obtained by our algorithm and *BestDatabasePartition* [7] on the multi-databases $T_{1,1}$,…, $T_{1,10}$ and $T_{4,1}$,…, $T_{4,10}$ for α=0.03 to 0.06

Multiple databases	# of candidate classification (m)	minsupp (α)	Best classification	goodness	Execution time (s)	
					BestDatabase-Partition	Proposed algorithm
$T_{1,1}$,…,$T_{1,10}$	45	0.03	{$T_{1,5}$, $T_{1,9}$, $T_{1,10}$} {$T_{1,8}$} {$T_{1,4}$, $T_{1,7}$} {$T_{1,6}$} {$T_{1,3}$} {$T_{1,2}$} {$T_{1,1}$}	4.75	0.00784	0.00227
		0.04	{$T_{1,10}$} {$T_{1,5}$, $T_{1,9}$} {$T_{1,8}$} {$T_{1,7}$} {$T_{1,6}$} {$T_{1,4}$} {$T_{1,3}$} {$T_{1,2}$} {$T_{1,1}$}	0.096	0.00804	0.00240
		0.05	{$T_{1,10}$} {$T_{1,9}$} {$T_{1,8}$} {$T_{1,3}$, $T_{1,5}$, $T_{1,7}$} {$T_{1,6}$} {$T_{1,4}$} {$T_{1,2}$} {$T_{1,1}$}	3.99	0.00799	0.00233
		0.06	{$T_{1,2}$, $T_{1,3}$, $T_{1,4}$, $T_{1,5}$, $T_{1,6}$, $T_{1,7}$, $T_{1,8}$, $T_{1,9}$, $T_{1,10}$} {$T_{1,1}$}	34.71	0.00777	0.00225
$T_{4,1}$,…,$T_{4,10}$	45	0.03	{$T_{4,4}$, $T_{4,10}$} {$T_{4,9}$} {$T_{4,8}$} {$T_{4,7}$} {$T_{4,6}$} {$T_{4,5}$} {$T_{4,3}$} {$T_{4,2}$} {$T_{4,1}$}	2.88	0.00780	0.00228
		0.04	{$T_{4,10}$} {$T_{4,5}$, $T_{4,9}$} {$T_{4,8}$} {$T_{4,7}$} {$T_{4,6}$} {$T_{4,4}$} {$T_{4,3}$} {$T_{4,2}$} {$T_{4,1}$}	4.44	0.00769	0.00217
		0.05	{$T_{4,10}$} {$T_{4,3}$, $T_{4,9}$} {$T_{4,8}$} {$T_{4,7}$} {$T_{4,6}$} {$T_{4,5}$} {$T_{4,4}$} {$T_{4,2}$} {$T_{4,1}$}	4.82	0.00766	0.00216
		0.06	{$T_{4,10}$} {$T_{4,9}$} {$T_{4,8}$} {$T_{4,7}$} {$T_{4,6}$} {$T_{4,3}$, $T_{4,5}$} {$T_{4,4}$} {$T_{4,2}$} {$T_{4,1}$}	4.62	0.00773	0.00218

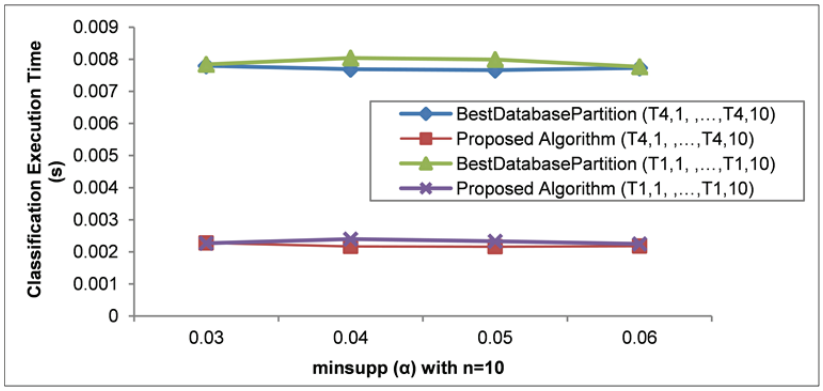


Fig. 3. Execution time versus minsupp (α).

Table 6. A comparison between the classification results obtained by our algorithm and *BestDatabasePartition* [7] on the multi-databases $T_{1,3},\ldots, T_{1,n}$ and $T_{4,3},\ldots, T_{4,n}$ under α=0.03 for n=3 to 20

Multiple databases (i=1,4)	# of multiple databases (n)	# of candidate classifications (m) from		goodness value of the best classification		Time complexity			
						Proposed algorithm $O(n^2)$		BestDatabasePartition $O(m\times n^2)$	
		$T_{1,3},\ldots, T_{1,n}$	$T_{4,3},\ldots, T_{4,n}$	$T_{1,3},\ldots, T_{1,n}$	$T_{4,3},\ldots, T_{4,n}$	$T_{1,3},\ldots, T_{1,n}$	$T_{4,3},\ldots, T_{4,n}$	$T_{1,3},\ldots, T_{1,n}$	$T_{4,3},\ldots, T_{4,n}$
$T_{i,3},\ldots, T_{i,n}$	3	3		0.67	0.80	9		27	
	4	6		0.77	1.30	16		96	
	5	10		0.59	1.63	25		250	
	6	15		1.46	1.80	36		540	
	7	21		2.11	1.78	49		1029	
	8	28		3.09	1.60	64		1792	
	9	36		5.91	1.32	81		2916	
	10	45		5.88	2.47	100		4500	
	11	55		7.73	1.45	121		6655	
	12	66		8.03	2.3	144		9504	
	13	78		10.28	3.35	169		13182	
	14	91		12.54	4.58	196		17836	
	15	105		15.57	6.01	225		23625	
	16	120	119	18.29	7.43	256		30720	30464
	17	136	135	21.31	9.05	289		39304	39015
	18	153	152	24.37	9.10	324		49572	49248
	19	171	170	28.27	11.13	361		61731	61370
	20	190	189	32.17	14.90	400		76000	75600

In the second part of our experiments, we studied how the number of databases n and the number of candidate classification m influence the time complexity of the two algorithms. Hence, we set the value of α=0.03 and we varied n from 3 to 20 databases. The experiment results obtained by the two algorithms are presented in Table 6 and Fig. 4. From the results, we can see that the time complexity and the execution time tend go higher as the value of n increases. However, we notice that the execution time increases rapidly for *BestDatabasePartition*. The reason is that m becomes larger as the value of n increases, as it is shown in Table 6. Since the time complexity of *BestDatabasePartition* depends strongly on m, we note a rapid increase of *BestDatabasePartition*'s execution time for n= 3 to 20.

From the experiment results above, we observe that our algorithm and *BestDatabasePartition* [7] find the same best classification of the n multiple databases. The reason is that the same similarity and goodness measures proposed in [7] have been used for both algorithms. However, the execution time of our algorithm is the shortest. This is because, unlike the existing classification algorithms [5-8], the time complexity of the classification generation procedure in our algorithm depends only on the number of databases (n). In the worst case, n^2 edges are processed to find the best classification of a set of n multiple databases.

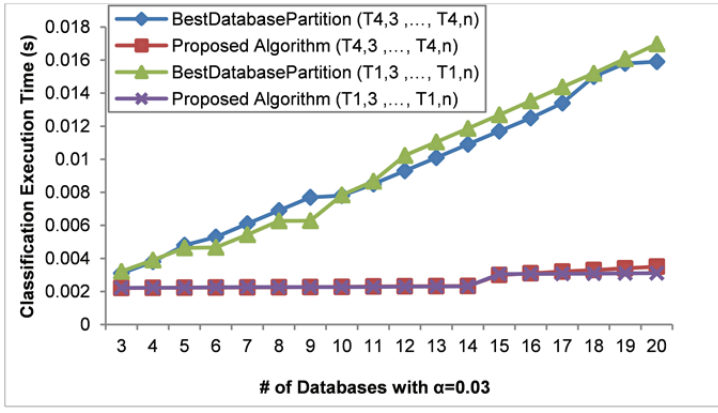


Fig. 4. Execution time versus the number of databases (n).

5. Conclusion

In this paper, we have proposed an improved database classification algorithm for multi-database mining. Different from the existing works, our algorithm searches for the best classification of a database set D by analyzing and determining the connected components of an undirected weighted graph $G=(D,E)$. The experiments have proved the accuracy and efficiency of our approach and the execution time of our algorithm is the shortest.

Future work will be directed toward improving the accuracy of the similarity measure between the databases. Also, we will assess our algorithm using real-world datasets and validate the tests on a real multi-databases system.

References

[1] S. Zhang and M. J. Zaki, "Mining multiple data sources: local pattern analysis," *Data Mining and Knowledge Discovery*, vol. 12, no. 2-3, pp. 121-125, 2006.

[2] S. Zhang, X. Wu, and C. Zhang, "Multi-database mining," *IEEE Computational Intelligence Bulletin*, vol. 2, no. 1, pp. 5-13, 2003.

[3] S. Zhang, C. Zhang, and X. Wu, *Knowledge Discovery in Multiple Databases*. New York, NY: Springer, 2004.

[4] A. Adhikari, P. Ramachandrarao, and W. Pedrycz, *Developing Multi-database Mining Applications*. London: Springer, 2010.

[5] X. Wu, C. Zhang, and S. Zhang, "Database classification for multi-database mining," *Information Systems*, vol. 30, no. 1, pp. 71-88, 2005.
[6] H. Li, X. Hu, and Y. Zhang, "An improved database classification algorithm for multi-database mining," in *Frontiers in Algorithmics*. Heidelberg: Springer, 2009, pp. 346-357.
[7] A. Adhikari and P. R. Rao, "Efficient clustering of databases induced by local patterns," *Decision Support Systems*, vol. 44, no. 4, pp. 925-943, 2008.
[8] Y. Liu, D. Yuan, and Y. Cuan, "Completely clustering for multi-databases mining," *Journal of Computational Information Systems*, vol. 9, no. 16, pp. 6595-6602, 2013.
[9] H. Liu, H. Lu, and J. Yao, "Identifying relevant databases for multidatabase mining," in *Research and Development in Knowledge Discovery and Data Mining*. Heidelberg: Springer, 1998, pp. 15-18.
[10] H. Liu, H. Lu, and J. Yao, "Toward multi-database mining: identifying relevant databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 4, pp. 541-553, 2001.
[11] R. Agrawal and J. C. Shafer, "Parallel mining of association rules," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 962-969, 1996.
[12] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago de Chile, Chile, 1994, pp. 487-499.
[13] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: a frequent-pattern tree approach," *Data Mining and Knowledge Discovery*, vol. 8, no. 1, pp. 53-87, 2004.
[14] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.

Salim Miloudi https://orcid.org/0000-0001-5543-7953

He received a M.S. degree in information systems engineering from the University of Science and Technology–Mohamed Boudiaf (USTOMB), Oran, Algeria in 2011. His research interests include data mining for knowledge discovery, multi-database mining and machine learning.

Sid Ahmed Rahal https://orcid.org/0000-0003-3230-5792

He is a Doctor in computer science since 1989 in Pau University France. He is a member in many professional activities: member of the Signal, Systems and Data Laboratory (LSSD). Research interests include the databases, data mining, agent and expert systems.

Salim Khiat https://orcid.org/0000-0002-8075-4536

He received a Ph.D. degree in computer science from the university of science and technology–Mohamed Boudiaf Oran USTOMB Algeria in 2015. He is a member of the Signal, Systems and Data Laboratory (LSSD). His research interests include the databases, multi-database mining, ontology, grid and cloud computing.