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 \(\frac{n^2-n}{2}\) 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.
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 $m$ ($1 \leq m \leq (n^2 - n)/2$) 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 $\delta$ 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 $\delta$. 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 \( m \)
\((1 \leq m \leq n^2)\) 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 classifyng 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 \( \text{class}(D, \delta_i) \) and \( \text{class}(D, \delta_j) \) be two candidate
classifications of \( D = \{D_1, D_2, \ldots, D_n\} \) generated at two consecutive similarity level \( \delta_i \) and \( \delta_j \).Then for each
cluster \( g_x \) in \( \text{class}(D, \text{sim}, \delta_i) \), there is certainly a cluster \( g_y \) in \( \text{class}(D, \text{sim}, \delta_j) \), 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 \( \text{class}(D, \text{sim}, \delta) \) at a similarity level \( \delta \), 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 \( \text{sim}(D_i, D_j) \). At a certain similarity level \( \delta \), there is an edge connecting two vertices \( D_i \) and \( D_j \) if and only if, \( \text{sim}(D_i, D_j) \geq \delta \).

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.

![Proposed approach for classifying the \( n \) multiple databases.](image)

**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 $E_{\delta_i} = \{(D_i, D_j) \in E \mid \text{sim}(D_i, D_j) = \delta_i, i,j=1 \ldots n, i \neq j\}$ be the list of edges with weight value equal to $\delta_i$ ($l=1$ to $m$).

For each distinct similarity value $\delta_i$, a candidate classification $\text{class}(D, \text{sim}, \delta_i)$ is generated by adding the edge list $E_{\delta_i}$ to the edge set $E$ such that $E = E_{\delta_1} \cup E_{\delta_2} \cup \ldots \cup E_{\delta_m}$ (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 $\text{class}(D, \text{sim}, \delta_i)$.

At a given similarity level $\delta_i$, if each connected component of $G$, denoted $g^{\delta_i}_{k}$, forms a clique (i.e., a subset of vertices, each pair of which is connected by an edge in $E$), then the corresponding classification $\text{class}(D, \text{sim}, \delta_i) = \{g^{\delta_i}_{1}, g^{\delta_i}_{2}, \ldots, g^{\delta_i}_{k}\}$ 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 $\text{sim}$ between two databases $D_i$ and $D_j$ is calculated as follows.

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

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

**Property 1.** $\text{sim}(D_i, D_j, \alpha)$ satisfies that:

1. $0 \leq \text{sim}(D_i, D_j, \alpha) \leq 1$;
2. $\text{sim}(D_i, D_i, \alpha) = \text{sim}(D_j, D_j, \alpha)$;
3. $\text{sim}(D_i, D_j, \alpha) = 1$ for $i,j=1\ldots n$

**Definition 2.** The distance measure $\text{dist}$ between databases $D_i$ and $D_j$ is defined as follows.

$$
\text{dist}(D_i, D_j, \alpha) = 1 - \text{sim}(D_i, D_j, \alpha)
$$

The distance function $\text{dist}$ has the following properties.

**Property 2.** $\text{dist}(D_i, D_j, \alpha)$ satisfies that:

1. $0 \leq \text{dist}(D_i, D_j, \alpha) \leq 1$;
2. $\text{dist}(D_i, D_j, \alpha) = \text{dist}(D_j, D_i, \alpha)$;
3. $\text{dist}(D_i, D_j, \alpha) = 1$ for $i,j=1\ldots n$

**Definition 3.** Let $\text{class}(D, \text{sim}, \delta) = \{g^{\delta_i}_{1}, g^{\delta_i}_{2}, \ldots, g^{\delta_i}_{k}\}$ be a classification of $k$ clusters of the database set $D = \{D_1, D_2, \ldots, D_n\}$ at the similarity level $\delta$. The intra-cluster similarity of $\text{class}(D, \text{sim}, \delta)$ is defined as follows.
intra-sim(class(D,δ)) = \sum_{i=1}^{k} \sum_{D_i \in \mathcal{G}^\delta} \text{sim}(D_i, D_j, \alpha) \tag{3}

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

\[\text{inter-dist}(\text{class}(D,\delta)) = \sum_{\overset{\alpha \in \mathcal{G}^\delta}{\alpha \neq \beta}} \sum_{D_i \in \mathcal{G}^\beta \neq D_j \in \mathcal{G}^\beta} \text{dist}(D_i, D_j, \alpha) \tag{4}\]

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,δ) is defined as follows.

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

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 \( D=\{D_i, D_2, \ldots, D_n\} \) 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 \( \delta \), a classification class(D,sim,δ) is complete if the following equation is verified:

\[\sum_{i=1}^{k} \frac{|\mathcal{G}^\delta_i|^2 - |\mathcal{G}^\delta_i|}{2} = |E| \tag{6}\]

Where \(|\mathcal{G}^\delta_i|\) denotes the number of vertices in the connected component \(\mathcal{G}^\delta_i\) and \(|E|\) is the number of edges in the graph \( G \). In other words, if each class \( \mathcal{G}^\delta_i \) \((i=1\) to \( k)\) in class(D,sim,δ) is a clique in \( G \) then class(D,sim,δ) 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>
<thead>
<tr>
<th>sim</th>
<th>D_1</th>
<th>D_2</th>
<th>D_3</th>
<th>D_4</th>
<th>D_5</th>
<th>D_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_1</td>
<td>1</td>
<td>0.40</td>
<td>0.56</td>
<td>0.40</td>
<td>0.50</td>
<td>0.40</td>
</tr>
<tr>
<td>D_2</td>
<td>-</td>
<td>1</td>
<td>0.40</td>
<td>0.79</td>
<td>0.40</td>
<td>0.45</td>
</tr>
<tr>
<td>D_3</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0.40</td>
<td>0.45</td>
<td>0.40</td>
</tr>
<tr>
<td>D_4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0.40</td>
<td>0.45</td>
</tr>
<tr>
<td>D_5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0.40</td>
</tr>
<tr>
<td>D_6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>
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 $\text{class}(D)=[\{D_1\}, \{D_2\}, \{D_3\}, \{D_4\}, \{D_5\}, \{D_6\}]$, with $\text{goodness}(\text{class}(D))=2.20$.

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

For $\delta=0.79$, we have, $E_{0.79}=[(D_2, D_4)]$, $E=E \cup E_{0.79}$ $\text{class}(D,\text{sim},0.79)=[\{D_2, D_4\}, \{D_1, D_3\}, \{D_6\}]$, which is a complete classification because:

$$\sum_{i=1}^{E} |\theta_{i, 0.79}^2 - |\theta_{i, 0.79}^2| = \left(\frac{||D_2||^2-||D_2, D_4||^2}{2} + \frac{||D_1||^2-||D_1||^2}{2} + \frac{||D_3||^2-||D_3||^2}{2} + \frac{||D_5||^2-||D_5||^2}{2} + \frac{||D_6||^2-||D_6||^2}{2}\right) = 1 = |E|$$

with $\text{goodness}(\text{class}(D,0.79))=3.78$

The remaining classifications are summarized in Table 2.

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

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

<table>
<thead>
<tr>
<th>Similarity level ($\delta$)</th>
<th>Edge set ($E_\delta$)</th>
<th>Candidate classification $\text{class}(D,\text{sim},\delta)$</th>
<th>$\text{goodness}(\text{class}(D,\delta))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.56</td>
<td>(D_1, D_3)</td>
<td>[D_2, D_4], [D_1, D_3], [D_5], [D_6]</td>
<td>4.90</td>
</tr>
<tr>
<td>0.50</td>
<td>(D_1, D_3)</td>
<td>[D_2, D_4], [D_1, D_3, D_5], [D_6]</td>
<td>5.90</td>
</tr>
<tr>
<td>0.45</td>
<td>(D_2, D_4), (D_1, D_3), (D_6, D_5)</td>
<td>[D_2, D_4, D_6], [D_1, D_3, D_5]</td>
<td>6.60</td>
</tr>
<tr>
<td>0.40</td>
<td>(D_1, D_3), (D_1, D_4), (D_1, D_6), (D_2, D_4), (D_2, D_5), (D_3, D_4), (D_3, D_6), (D_4, D_5), (D_5, D_6)</td>
<td>[D_1, D_2, D_3, D_4, D_5, D_6]</td>
<td>5.80</td>
</tr>
</tbody>
</table>

### 3.2 Data Structure and Algorithms

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

1) Find the list of edges with weight value equal to $\delta$, denoted $E_\delta$, 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 $E_\delta$ in the decreasing order of the weight $\delta$. 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 $E_i$. 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 \(E_\delta\)

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 \(\delta\) and the index of the first edge in \(E_\delta\) respectively. Edges with the same weight value \(\delta\) 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 \(E_\delta\).

In the following procedure, we present the algorithm used to construct the ordered edge lists \(E_\delta\).

**Procedure Build _Edge_Tree\((n, M)\)**

**Input:** 
- \(n\): the number of databases; \(M[n\times n]\): 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 \(
\{D_1\}, \{D_2\}, \ldots, \{D_n\}\)

1. **let** \(dist\_inter=0\);
2. **let** \(T\.root=null\)
3. **let** \(k=1;\) // \(k\) is the index of the current edge
4. **for** \(i=1\) to \(n-1\), \(j=i+1\) to \(n\) **do**
5. **let** \(\delta =M[i,j]\) //get the similarity between \(D_i\) and \(D_j\)
6. **let** \(dist\_inter = dist\_inter + (1-\delta)\);
7. **let** \(V[k]=-1\);
8. **let** \(x=Tree\_Search(T\.root, \delta)\);
9. **if** \(x\# null\) and \(x\.weight=\delta\) **then**
10. **let** \(V[k]=x\.index\);
11. **let** \(x\.index=k;\) //Add the \(k\)-th edge to the list \(E_\delta\)
12. **else**
13. **Allocate** a new node object \(y\);
14. **let** \(y\.weight=\delta\);
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 \(\delta\) exists in \(T\). If there is a node \(x\) in \(T\) such that \(x\.weight=\delta\) then the current edge (i.e., the \(k\)-th edge) is inserted at the beginning of \(E_\delta\) 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 \(\delta\) and \(y\.index\) 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 \( \delta \) a new node is inserted into \( T \) if there is no node containing the weight \( \delta \). 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.

![Fig. 2. The balanced binary search tree \( T \) and the index array \( V \) (the column index) of the example.](image)

Each node of \( T \) contains mainly 4 fields (the similarity value \( \delta \), the index of the first edge in the list \( E_\delta \), 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:

\[
i = n - h \\
j = (i + 1) + h(h + 1)/2 - (C_n^{k+1} - k + 1)
\]

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

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

**Proof.** Let \( M \) be the triangular similarity table of size \( n \times n \). Then, there are \( C_n^h \) similarity values in \( M \), which are associated with \( C_n^h \) edges indexed from 1 to \( C_n^h \). 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 \( C_n^{k+1} - k + 1 \) 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: $(h + 1/2)^2 \geq (8(C_n^2 - k + 1) + 1)/4 \Rightarrow h \geq \left(\sqrt{8 \times (C_n^2 - k + 1) + 1} - 1\right)/2$. Therefore, $i = n - h$ such that $h = \left\lceil \sqrt{8 \times (C_n^2 - k + 1) + 1} - 1\right\rceil/2$. 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 of using pointers, we use integers to link child nodes to their parent nodes. For example, if $x_i$ is the parent of $x_o$, then we set $x_o.parent=j$. 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 $i$-th vertex, **findset**($x_i$) compresses the path between $x_i$ and the root when $x_i$ and $x_j$’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($D,sim,\delta$) : 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 (Di, Dj) 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 if
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
42. 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: $x_i$, $x_j$: two disjoint components to merge
- class_version: the version of the current classification
- best_class_version: the version of the best classification

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

Procedure findset($x_i$, class_version, best_class_version)

Input: $A[1..n]$: the array representing the $n$ nodes of the disjoint-set forest
- $x_i$: 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 $x_i$.parent = −1 then return $x_i$;
02. else if $A[x_i$.parent].parent = −1 then return $A[x_i$.parent];
03. else
04. if $x_i$.version ≤ best_class_version then let $x_i$.parent_temp = $x_i$.parent; end if;
05. let $x_i$.parent = $A[x_i$.parent].parent;
06. let $x_i$.version = class_version; // update the current parent version $x_i$.version;
07. return $A[x_i$.parent];
08. end if;

Discussion. Initially, the graph $G=(D,E)$ has $n$ disjoint graph components. Therefore, we get a trivial classification $\text{class}(D, \text{sim})=\{(D_1), (D_2), ..., (D_n)\}$. Lines 1-7 initialize the variables used in our algorithm. The inter-cluster distance ($\text{dist}_{\text{inter}}$) between the $n$ components is calculated during the call of the procedure BuildEdgeTree. The intra-cluster similarity ($\text{sim}_{\text{intra}}$) has a zero value since there is one database in each cluster. We use the variable $\text{goodness}_{\text{max}}$ 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 $E_\delta$. 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 $E_\delta$ such that $\delta = e$.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 (Di,Dj) 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 (Di,Dj) to find the connected components to which Di and Dj belong. The time complexity of lines 17-18 is O(1).

Let xi and xj be the nodes returned by findset. At line 19, if xi and xj 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 Di and Dj 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 Eδ from the index array V[1..(n²−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) ≥ 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≤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 xi (1 ≤ i ≤ n) at the time that the best classification was generated. That is, if x.version≤best_class_version, then x.parent is returned as the cluster containing the i-th database. Otherwise, x.parent_temp is returned instead.

The algorithm terminates once all the m edge lists Eδ are examined and the m candidate classifications are generated. The average size of each edge list is n²/m. Therefore, exploring all the m edge lists takes O(n²) 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(n²) time. Our algorithm is optimal when comparing with BestDatabasePartition [7], which takes O(m×n²) 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
\[ m \times (2 \text{pointers} + \text{real} + \text{integer}) \text{ units}. \] 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 \( n \times (4 \text{integer}) \text{ units} \). Therefore, the memory space required by our algorithm is \( (2n^2+14n+28m) \text{ bytes} \) 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, \textit{BestDatabasePartition} needs \( n^2 \) real units to store the similarity relation table. Assuming that \textit{BestDatabasePartition} uses a balanced binary search tree to sort all the distinct similarity values, then \( m \times (2 \text{pointers} + \text{real}) \text{ units} \) will be required to store the nodes of the tree. Each candidate classification is stored separately. Hence, \( mn \times \text{integer units} \) are required to store all the \( m \) candidate classifications. Therefore, the memory space required by \textit{BestDatabasePartition} is \( (8n^2+20m+4mn) \text{ bytes} \) 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 \textit{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 \textit{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 \( T_{1,1} \), \( T_{1,2} \), \ldots \( \), \( T_{1,n} \) and \( T_{4,1} \), \( T_{4,2} \), \ldots \( \), \( T_{4,n} \) respectively, such that \( n=10, 20 \).

<table>
<thead>
<tr>
<th>Table 3. Synthetic datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
</tr>
<tr>
<td>T10I4D100K</td>
</tr>
<tr>
<td>T40I10D100K</td>
</tr>
</tbody>
</table>

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

Once the data-preparation is done, we classify the multi-databases using our algorithm and \textit{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 \text{minsupp}(\alpha) \) on the classification execution time. Therefore, we set the number of databases \( n=10 \) and we varied the value of \( \alpha \) to get
different sets of frequent itemsets. Then, we executed our algorithm and BestDatabasePartition on the same frequent 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 \( \alpha \) 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 \( \alpha=0.03 \) to 0.06.

Table 4. The sets of frequent itemsets from the multiple databases \( T_{1,1}, T_{1,2}, \ldots, T_{1,n} \) and \( T_{4,1}, T_{4,2}, \ldots, T_{4,n} \) for \( \alpha=0.03 \) to 0.06 and \( n=10, 20 \)

<table>
<thead>
<tr>
<th>Dataset ((i=1,4))</th>
<th># of multiple databases ((n))</th>
<th># of transactions in each database</th>
<th>Algorithm</th>
<th>Minimum support ((\alpha))</th>
<th>Sets of frequent itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_i )</td>
<td>10</td>
<td>10,000 (( \times 10))</td>
<td>FP-Growth</td>
<td>0.03</td>
<td>( \text{FIS}(T_{i,1}, 0.03), \text{FIS}(T_{i,2}, 0.03), \ldots, \text{FIS}(T_{i,10}, 0.03) )</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>4,500 (( \times 20))</td>
<td></td>
<td>0.03</td>
<td>( \text{FIS}(T_{i,1}, 0.03), \text{FIS}(T_{i,2}, 0.03), \ldots, \text{FIS}(T_{i,20}, 0.03) )</td>
</tr>
</tbody>
</table>

Table 5. A comparison between the classification results obtained by our algorithm and BestDatabasePartition [7] on the multi-databases \( T_{1,1}, \ldots, T_{1,10} \) and \( T_{4,1}, \ldots, T_{4,10} \) for \( \alpha=0.03 \) to 0.06

<table>
<thead>
<tr>
<th>Multiple databases</th>
<th># of candidate classification ((m))</th>
<th>minsupp ((\alpha))</th>
<th>Best classification</th>
<th>goodness</th>
<th>Execution time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{1,1}, \ldots, T_{1,10} )</td>
<td>45</td>
<td>0.03</td>
<td>{T_{1,5}, T_{1,9}, T_{1,10}}, {T_{1,8}}</td>
<td>4.75</td>
<td>0.00784, 0.00227</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.04</td>
<td>{T_{1,10}}, {T_{1,5}, T_{1,9}}, {T_{1,8}}, {T_{1,7}}</td>
<td>0.996</td>
<td>0.00804, 0.00240</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.05</td>
<td>{T_{1,10}}, {T_{1,9}}, {T_{1,8}}, {T_{1,7}}, {T_{1,6}}, {T_{1,5}}, {T_{1,4}}, {T_{1,3}}, {T_{1,2}}, {T_{1,1}}</td>
<td>3.99</td>
<td>0.00799, 0.00233</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.06</td>
<td>{T_{1,10}}, {T_{1,9}}, {T_{1,8}}, {T_{1,7}}, {T_{1,6}}, {T_{1,5}}, {T_{1,4}}, {T_{1,3}}, {T_{1,2}}, {T_{1,1}}</td>
<td>34.71</td>
<td>0.00777, 0.00225</td>
</tr>
<tr>
<td>( T_{4,1}, \ldots, T_{4,10} )</td>
<td>45</td>
<td>0.03</td>
<td>{T_{4,6}, T_{4,10}}, {T_{4,5}}, {T_{4,4}}, {T_{4,3}}, {T_{4,2}}, {T_{4,1}}</td>
<td>2.88</td>
<td>0.00780, 0.00228</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.04</td>
<td>{T_{4,10}}, {T_{4,5}, T_{4,9}}, {T_{4,8}}, {T_{4,7}}, {T_{4,6}}, {T_{4,5}}, {T_{4,4}}, {T_{4,3}}, {T_{4,2}}, {T_{4,1}}</td>
<td>4.44</td>
<td>0.00769, 0.00217</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.05</td>
<td>{T_{4,10}}, {T_{4,9}}, {T_{4,8}}, {T_{4,7}}, {T_{4,6}}, {T_{4,5}}, {T_{4,4}}, {T_{4,3}}, {T_{4,2}}, {T_{4,1}}</td>
<td>4.82</td>
<td>0.00766, 0.00216</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.06</td>
<td>{T_{4,10}}, {T_{4,9}}, {T_{4,8}}, {T_{4,7}}, {T_{4,6}}, {T_{4,5}}, {T_{4,4}}, {T_{4,3}}, {T_{4,2}}, {T_{4,1}}</td>
<td>4.62</td>
<td>0.00773, 0.00218</td>
</tr>
</tbody>
</table>
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 \( \alpha=0.03 \) 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 to go higher as the value of \( n \) increases. However, we notice that the execution time increases rapidly for \( \text{BestDatabasePartition} \). 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 \( \text{BestDatabasePartition} \) depends strongly on \( m \), we note a rapid increase of \( \text{BestDatabasePartition} \)'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*).](image_url)

### 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


Contribution to Improve Database Classification Algorithms for Multi-Database Mining


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.