الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي و البحث العلمي

جامعة وهران للعلوم و التكنولوجيا محمد بوضياف

# THÈSE

## En vue de l'obtention du Diplôme de Doctorat

***Présentée par :***

***Mr MILOUDI SALIM***

***Intitulée***

# *Le clustering pour la fouille multi-sources de données*

***Faculté*** : ***Mathématiques et Informatique***

***Département*** : ***Informatique***

***Domaine*** : ***MI***

***Filière*** : ***Informatique***

***Intitulé de la Formation*** : ***Ingénierie des Systèmes d'Information***

***Devant le Jury Composé de :***

| *Membres du Jury* | *Grade* | *Qualité* | *Domiciliation* |
|---|---|---|---|
| *Mme Fatima Bendella* | *Professeur* | *Président* | *USTO-MB* |
| *Mr Rahal Sid Ahmed Hebri* | *Professeur* | *Encadrant* | *USTO-MB* |
| *Mme Zaoui Lynda* | *Professeur* | *Examinateur* | *USTO-MB* |
| *Mr Belalem Ghalem* | *Professeur* | *Examinateur* | *Univ. Oran 1* |
| *Mr Ghomari Abdelghani* | *Maître de conférence-A* | *Examinateur* | *Univ. Oran 1* |
| *Mr Khiat Salim* | *Maître de conférence-B* | *Invité* | *ENPO-ORAN* |

***Année Universitaire : 2018 / 2019***

# *Remerciements*

*En tout premier lieu, je remercie Allah dieu tout puissant pour la capacité, le courage et la volonté qui m'ont été accordés pour compléter ce modeste travail de recherche.*

*Permettez-moi d'exprimer mes remerciements les plus chaleureux à mes superviseurs Pr. Rahal Sid Ahmed et Dr. Khiat Salim de m'avoir guidé et conseillé avec patience tout au long de la préparation de ma thèse.*

*Je tiens à remercier aussi l'ensemble des membres du Jury qui me font honneur d'avoir accepté de juger ce travail.*

*Je voudrais aussi saisir cette opportunité pour présenter en quelques mots ma profonde gratitude à tous ceux qui m'ont soutenu durant ma recherche.*

*En fin, je dédie ce travail à ma très chère famille, merci pour leur soutien et leurs encouragements et de m'avoir fourni l'atmosphère idéale pour compléter ce travail.*

***Miloudi Salim.***

# *Résumé*

L'apparition des entreprises multi-branches a conduit à la nécessité d'analyse des multi-sources de données distribuées à travers les différentes succursales de ces organisations.

Pour des fins de prise de décision, les entreprises multi-branches ont besoin d'explorer leurs multi-sources de données dans le but d'extraire des connaissances appelées motifs. Les techniques traditionnelles de fouille de données intègrent ces multi-sources en un seul ensemble de données pour la découverte des motifs globaux. Cependant, cette approche peut engendrer un coût de recherche important pour un traitement centralisé et peut empêcher la découverte de nouveaux types de motifs.

Pour faire face à ces problèmes, il est important de classer ces multi-sources de données en clusters similaires et distincts les uns des autres. Ainsi, chaque cluster pourrait être analysé individuellement pour la découverte de nouveaux motifs reflétant des informations spécifiques à chaque groupe de succursale de l'organisation.

Les techniques de clustering existantes appliquées aux multi-sources de données génèrent plusieurs classifications hiérarchiques et imbriquées sans tenir compte de la réutilisation des clusters générés dans les niveaux précédents. Par conséquent, un coût additionnel est ajouté au coût total de recherche de la meilleure classification. De plus, la mesure de similarité utilisée par les algorithmes existants ne prend pas en charge la contribution des motifs locaux non fréquents. Ces derniers peuvent grandement contribuer dans la synthèse des motifs globaux utiles à toute l'organisation.

Dans cette thèse, nous proposons une approche de clustering basée sur la découverte des groupes de bases de données cohésifs et pertinents dans un graphe de similarité. La particularité de ce graphe est qu'il permet de garder une trace des clusters générés dans les classifications précédentes pour les réutiliser ultérieurement. L'approche proposée permet aussi d'améliorer la précision de la mesure de similarité en faisant intervenir les motifs locaux non fréquents.

Afin d'évaluer la performance de notre algorithme, des expérimentations ont été menées sur des bases de données synthétiques en comparant les résultats obtenus avec ceux de la littérature. L'analyse des résultats expérimentaux démontre bien le potentiel de l'approche proposée à trouver la meilleure classification des multi-sources de données en un temps optimal.

***Mots-clefs :*** *Clustering, Mesure de similarité, Graphe complet, Composantes connexes, Fouille multi-bases de données, Motif locaux, Motif globaux.*

# *Abstract*

The emerging of multi-branch organizations has led to the need of analyzing the data coming from the multiple sources distributed through the different branches of these organizations.

For business decision making purpose, multi-branch companies need to mine their multiple data sources in order to extract useful information called patterns. Traditional data mining techniques integrate all these multi-sources into one huge dataset for pattern discovery. However, this approach may generate an expensive search cost for centralized processing and might disguise some important patterns.

To deal with the latter problems, it is important to classify these multiple databases into clusters of similar and relevant databases. Hence, each cluster could be analyzed individually to discover new patterns reflecting specific information about each group of branches of a given organization.

Existing clustering techniques applied to multi-data sources generate many nested hierarchical classifications independently, without taking into consideration the reuse of the clusters generated at the previous levels. Consequently, an overhead and useless cost is added up to the overall cost of searching the best classification for the multi-data sources. Moreover, the similarity measure used by the existing algorithms do not take into account the contribution of infrequent local patterns. The latter might largely contribute in synthesizing global patterns which are useful for the entire multi-branch organization.

In this thesis, we propose a clustering approach based on discovering cohesive and relevant database groups within a similarity graph which keeps track of the clusters generated previously while improving the accuracy of the similarity measure by including the participation of the infrequent local patterns.

In order to assess the performance, the running time and accuracy of the proposed clustering algorithm, experimentations on public databases have been conducted by comparing the obtained results with those of the existing algorithms. Analyzing the experimental results demonstrates the potential of the proposed approach to find the best classification of the multiple data sources within an optimal time.

***Keywords :*** *Clustering, Similarity measure, Complete graph, Connected components, Multi-database mining, Local patterns, Global patterns.*

# Table des Matières

# Partie II – Contribution

## Chapitre 3 -Amélioration de la performance du clustering des multi-bases de données

## Chapitre 4 -Expérimentations & Résultats

# Liste des figures

# Liste des tableaux



# Liste d'abréviation

**DDM :** Fouille de données distribuée.

**ECD (KDD) :** Extraction des connaissances à partir des données.

**FIS :** Ensemble d'itemset fréquents.

**FMBD :** Fouille multi-bases de données.

**JVM :** la machine virtuelle Java.

**RF:** Facteur de pertinence.

# Introduction générale

## 1 Problématique

L'utilisation croissante de la technologie des bases de données dans les entreprises multi-branches a conduit au développement des systèmes de fouille et d'analyse des multi-sources de données. En effet, le principe de ces systèmes consiste à fédérer différentes sources de données potentiellement hétérogènes pour donner l'illusion à l'utilisateur de l'existence d'une seule base de données homogène et centralisée.

Pour des fins de prise de décision, les grandes entreprises ont besoin d'explorer les multi-bases de données distribuées à travers leurs succursales afin d'extraire des connaissances utiles appelés *motifs*. Bien que la collection des sources de données multiples nous apporte des possibilités d'améliorer la qualité des décisions prises, il y a aussi d'importants défis à relever tels que comment extraire les connaissances utiles à partir des différentes sources de données et comment les intégrer. Nous appelons cela *le problème de la fouille multi-sources de données.*

Le processus traditionnel de la fouille multi-sources de données consiste à intégrer toutes les données brutes des différences sources en un seul ensemble de données pour la recherche des motifs utiles via les techniques de fouille de données telles la découverte des *itemsets fréquents* et des *règles d'association* [Agrawal et al. 1994] [Agrawal and Shafer 1996] [Han et al. 2000] [Park et al. 1995]. Cependant, cela peut accumuler une énorme base de données pour un traitement centralisé et peut générer aussi des problèmes sérieux liés à la confidentialité, l'incohérence et l'impertinence des données. En outre, ces techniques traditionnelles peuvent masquer les motifs utiles à cause de la grande quantité des données non pertinentes incluses dans le processus de fouille de données.

## 2 Motivations et contributions

Pour réduire le coût de la recherche lors de la découverte des motifs utiles dans les multi-bases de données, l'identification et la sélection des bases de données les plus pertinentes à une application de fouille de données [Liu et al. 2001] [Liu et al. 1998] représente un moyen efficace pour limiter l'espace de données à explorer et pour éviter de masquer les motifs utiles dans les multi-bases. Cette stratégie de classification des multi-bases de données connue aussi sous le nom de *sélection de base de données* dépend de l'objectif ou requête de l'utilisateur.

Lorsque l'application de fouille de données est à usage général où aucune requête utilisateur ne doit être spécifiée, la stratégie de sélection des bases de données devient insuffisante et non fonctionnelle. D'où la nécessité de développer une stratégie de *classification des bases de données* indépendante de toute application utilisateur. Ce processus de classification fait référence au *clustering des multi-bases de données* permettant de rechercher les groupes de base de données les plus similaires via l'utilisation une mesure de similarité appropriée. Une fois le processus achevé, les bases de données de chaque groupe peuvent être intégrées et analysées en utilisant les techniques traditionnelles de fouille de données. D'autres stratégies telles que l'*analyse des motifs locaux* [Zhang and Zaki 2006] peuvent aussi être appliquées pour la découverte de nouveaux types de motifs au niveau de chaque cluster de bases de données.

Quelques approches de clustering des multi-bases de données ont été conçues dans la littérature [Adhikari et al. 2010] [Li et al. 2009] [Liu et al. 2013] [Wu et al. 2005]. Ces approches diffèrent les unes des autres selon la mesure de similarité inter-bases de données utilisée, le temps d'exécution pris par l'algorithme de clustering et aussi dans l'évaluation de la meilleure classification. Ces différences permettent à ces approches de posséder des avantages et des limites qui nécessitent d'être discutés afin d'être améliorées.

Le développement des méthodes et techniques de clustering reste parmi les problèmes clés dans la conception des systèmes multi-bases de données. En effet, cela permettrait de réduire le coût de recherche dans les multi-bases et améliorer le processus décisionnel.

Les contributions que nous avons apportées dans cette thèse sont les suivantes :
La première contribution [Miloudi et al. 2018] vise à améliorer le temps d'exécution du clustering en proposant un algorithme basé sur la détermination des composantes connexes

dans un graphe de bases de données pondéré. La deuxième contribution [Miloudi et al. 2016a] consiste à améliorer la précision de la mesure de similarité en incluant le support des itemsets non fréquents.

Des expérimentations ont été réalisées sur des bases de données synthétiques et des matrices de similarité pré-calculées en comparant le temps d'exécution et le clustering généré de notre algorithme avec ceux de la littérature dans [Adhikari et al. 2010] [Li et al. 2009] [Liu et al. 2013] [Wu et al. 2005]. Les résultats obtenus prouvent bien l'efficacité de l'algorithme proposé à trouver la meilleure classification en un temps optimal.

## 3 Organisation de la thèse

Les chapitres de notre thèse sont organisés en deux parties comme suit :

**Partie I - Etat de l'art :** contient deux chapitres décrits comme suit :

- Dans le premier chapitre, nous introduisons la fouille multi-bases de données (FMBD) tout en argumentant son apparition en citant les limites de la FMBD traditionnelle. A la fin de ce chapitre, nous mettons la lumière sur l'importance du clustering des multi-bases de données comme étape de prétraitement des données dans le processus de FMBD.

- Dans le deuxième chapitre, nous présentons les approches de classification des multi-bases de données existantes ainsi que leurs algorithmes. Cette étude est conclue par une synthèse et étude critique des différentes approches étudiées. Les critères utilisés pour évaluer la performance des algorithmes de clustering sont donnés au début de ce chapitre.

**Partie II - Contribution :** organisée en deux chapitres comme suit :

- Dans le troisième chapitre nous présentons notre contribution pour l'amélioration de la performance du clustering des multi-sources de données.

- Les expérimentations réalisées et les résultats obtenus sont présentés et discutés dans le quatrième chapitre, tout en comparant la performance de l'algorithme proposé avec ceux existants dans la littérature.

Notre thèse se termine par une conclusion générale et les travaux futurs à réaliser pour la suite de notre recherche. Dans l’annexe, nous présentons d’autres algorithmes qui pourront être utilisés pour améliorer l’approche proposée encore plus en matière temps d’exécution.

# Partie I – Etat de l’art

# Chapitre 1 –

# La fouille multi-bases de données (FMBD)

# 1 Introduction

De nos jours, les organisations et les entreprises multi-branches dans les différents secteurs économique, sanitaire et éducationnel ont mis en œuvre des mesures visant à informatiser l'ensemble ou une partie de leurs fonctions quotidiennes. La plupart de ces organisations comprennent plusieurs succursales situées dans différentes régions géographiques et chaque succursale possède au moins une base de données locale.

Pour ces entreprises, il est important de manipuler les données se trouvant dans leurs succursales de façon rapide et fiable. Cependant, ce besoin est difficile à satisfaire lorsque les données sont stockées dans des bases de données locales et indépendantes, mais aussi d'une même importance à toute l'organisation.

Malgré ces difficultés, les chercheurs ont intensifié leurs efforts pour le développement des techniques de gestion et d'interrogation des collections de bases de données hétérogènes. Ainsi, le développement des systèmes multi-bases de données est devenu un domaine de recherche important au sein de la communauté de base de données.

D'autre part, pour des fins de prise de décision, il est primordial d'analyser les multi-bases de données afin d'en extraire des connaissances utiles à l'ensemble de toute l'organisation. Par conséquent, le développement des techniques de préparation et de fouille des multi-bases de données (FMBD) est à la fois un défi et une tâche cruciale.

Dans ce chapitre, nous allons étudier l'importance de développement des stratégies de FMBD afin de répondre aux besoins locaux et globaux des organisations multi-branches. En outre, nous allons présenter les techniques de fouille multi-bases de données, leurs limites et les défis à relever pour les surpasser.

# 2 Processus d'extraction de connaissance et fouille de données

Durant ces dernières années, la croissance des moyens de génération et de collection des données a induit à la nécessité de développement des moyens de stockage, d'interrogation et d'analyse de ces données.

Par ailleurs, pour une exploitation efficace de ces masses de données, il est important de développer des techniques et des méthodes intelligentes pour le prétraitement et l'extraction des informations utiles (appelées connaissances) à partir des bases de données volumineuses. C'est ainsi que le processus d'Extraction de Connaissances à partir des Données (ECD) est apparu.

En effet, « L'ECD est un processus semi-automatique et itératif constitué de plusieurs étapes allant de la sélection et la préparation des données jusqu'à l'interprétation des résultats, en passant par la phase d'extraction de connaissances appelée *fouille de données* (en anglais datamining) »[Khiat 2015]. Ce processus est décrit dans la figure 1.1

Dans l'ECD, la phase de *fouille de données* désigne l'application de méthodes et techniques aux données préparées et transformées afin d'extraire des modèles permettant de décrire les corrélations qui existent entre les différents items de données.

Par conséquent, la fouille de données est l'une des étapes les plus importantes dans l'ECD et elle est représentée par un ensemble d'algorithmes qu'on peut classer en trois catégories: Algorithmes de classification, de clustering et d'extraction d'itemsets fréquents et de règles d'association. Le choix d'un algorithme par rapport à un autre dépend de l'application de la fouille de données.

Avec l'émergence des entreprises multi-branches, le besoin d'extraire des connaissances à partir des multi-bases de données distribuées a fait son apparition. Ainsi, une extension de la fouille de données s'est imposée donnant naissance à un nouveau processus appelé *fouille multi-bases de données* (FMBD) qui est décrit dans la section suivante.

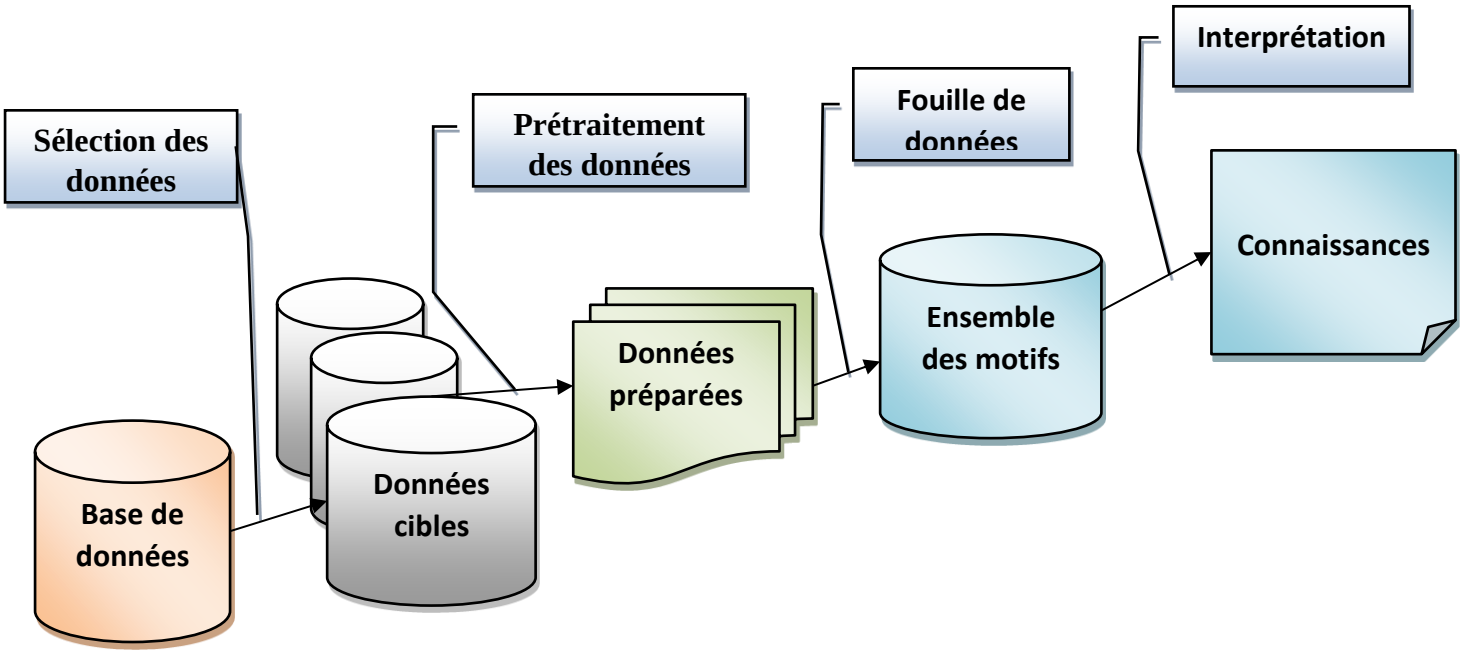



**FIGURE 1.1** – ETAPES DU PROCESSUS D'EXTRACTION DE CONNAISSANCES (ECD)

# 3 La fouille multi-bases de données

## 3.1 Définition

La fouille multi-bases de données traditionnelle connue aussi sous le nom de *la fouille monobase de données* consiste à intégrer toutes les données issues des bases locales d'une organisation pour la découverte des connaissances par les techniques de fouille monobase de données [Agrawal et al. 1994] [Agrawal and Shafer 1996] [Han et al. 2000] [Park et al. 1995]. Cependant, ces techniques sont inadéquates pour les applications permettant de prendre plusieurs types de décisions au sein des sociétés multi-branches.

En effet, dans une société multi-branches constituée de son siège central et de ses succursales, comme le montre la figure 1.2**,** on peut avoir besoin de deux types d'application : les applications locales et les applications globales.

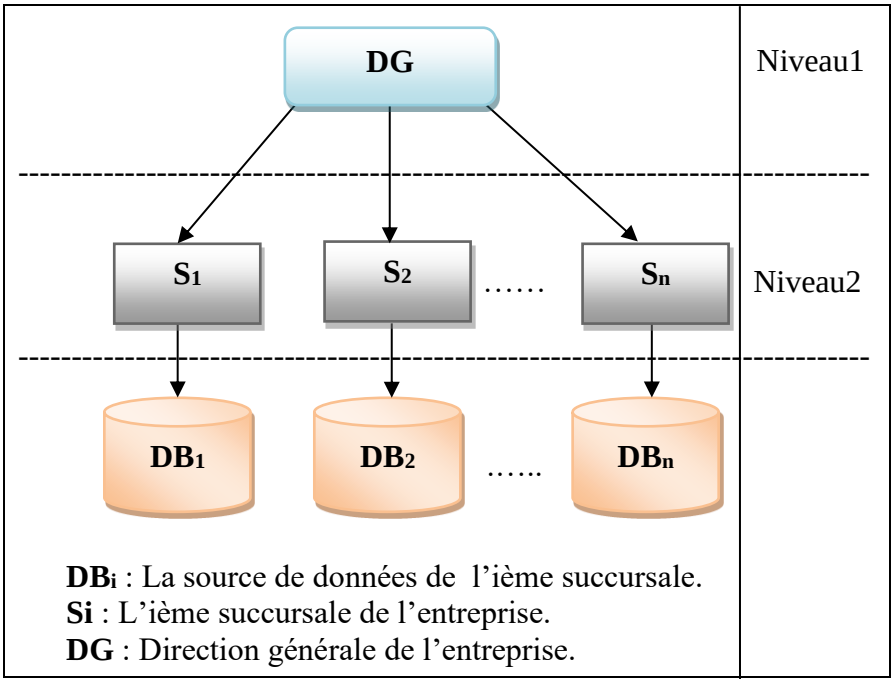



**FIGURE 1.2 -**ENTREPRISE MULTI-BRANCHES.

En utilisant les applications globales, le siège central de la société est plus intéressé par les motifs supportés ou pris en charge par la plupart de ses succursales afin de prendre des décisions globales sur l'ensemble des succursales et ces motifs sont appelés des *motifs globaux*, parfois aussi des *motifs majoritaires*.

De l'autre côté, en utilisant les applications locales, le directeur de la succursale a besoin d'explorer ces bases de données locales pour créer des politiques de gestions internes. Les applications à deux niveaux dans une société multi-branches sont représentées dans la figure 1.3.

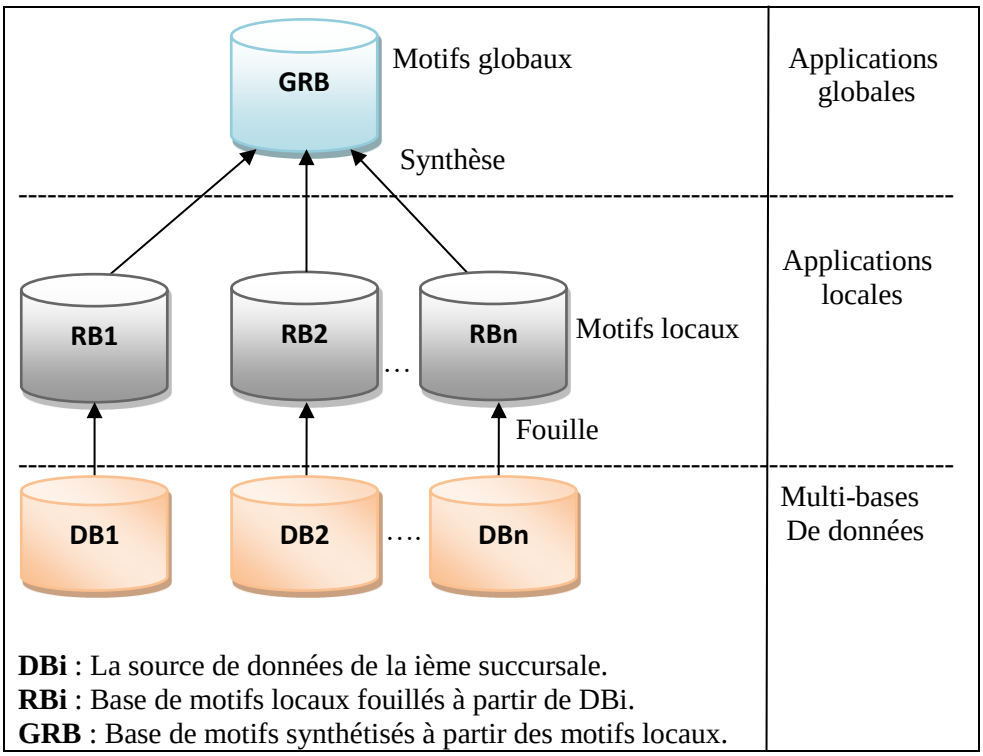



**FIGURE 1.3 -** APPLICATIONS A DEUX NIVEAUX DANS UNE SOCIETE MULTI-BRANCHES

La transmission des motifs locaux (plutôt que les lignes de données brutes) au siège central de la société centrale fournit un moyen efficace pour faire face aux problèmes liés aux multi-bases de données. Cette stratégie de fouille multi-bases de données fait référence à *l'analyse des motifs locaux* [Zhang and Zaki 2006]. De l'autre côté, le nombre de motifs locaux transmis peut être si grand que la découverte des motifs intéressants peut devenir difficile pour le siège central de la société. Par conséquent, il est nécessaire de développer des techniques efficaces pour l'identification des motifs potentiellement utiles dans les multi-bases de données.

### 3.2 Evolution du processus de la fouille multi-bases de données

Afin d'étudier la fouille multi-bases de données (FMBD), il est important de comprendre l'évolution de la fouille de données dans les multi-bases au fil des années. En effet, le processus traditionnel de la FMBD est illustré comme suit :

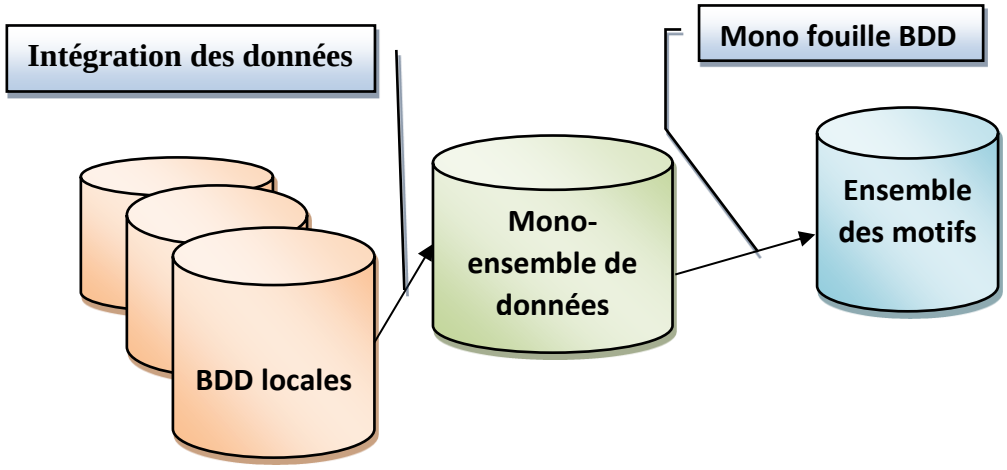



**FIGURE 1.4 -** PROCESSUS TRADITIONNEL DE LA FOUILLE MULTI-BASES DE DONNEES

- BDD locales : représentent l'ensemble des bases de données locales des *n* succursales d'une société multi-branches.
- Intégration des données : consiste à intégrer toutes les bases de données en une seule base appelée mono-ensemble de données.
- Mono-fouille BDD: consiste à utiliser les techniques de fouille de données proposées dans la littérature [Agrawal et al. 1994] [Agrawal and Shafer 1996] [Han et al. 2000] [Park et al. 1995] pour la découverte des motifs globaux dans le mono-ensemble de données.
- L'ensemble des motifs représente les itemsets fréquents découverts à partir du mono-ensemble de données.

Pour faire face à la taille des ensembles de données, [Liu et al. 2001] [Liu et al. 1998] ont proposé une technique qui permet de sélectionner uniquement les bases de données les plus pertinentes à une requête utilisateur avant de les intégrer. Cette approche est illustrée dans la figure 1.5. Une mesure *RF* (facteur de pertinence) a été proposée pour identifier les bases de données à sélectionner pour trouver des motifs utiles à la requête utilisateur.

Cela permet d'éviter la jointure forcée de toutes des bases dans le processus traditionnel de fouille de données. ¨De plus, cette approche est efficace pour réduire le coût de recherche des motifs dans les multi-bases de données.

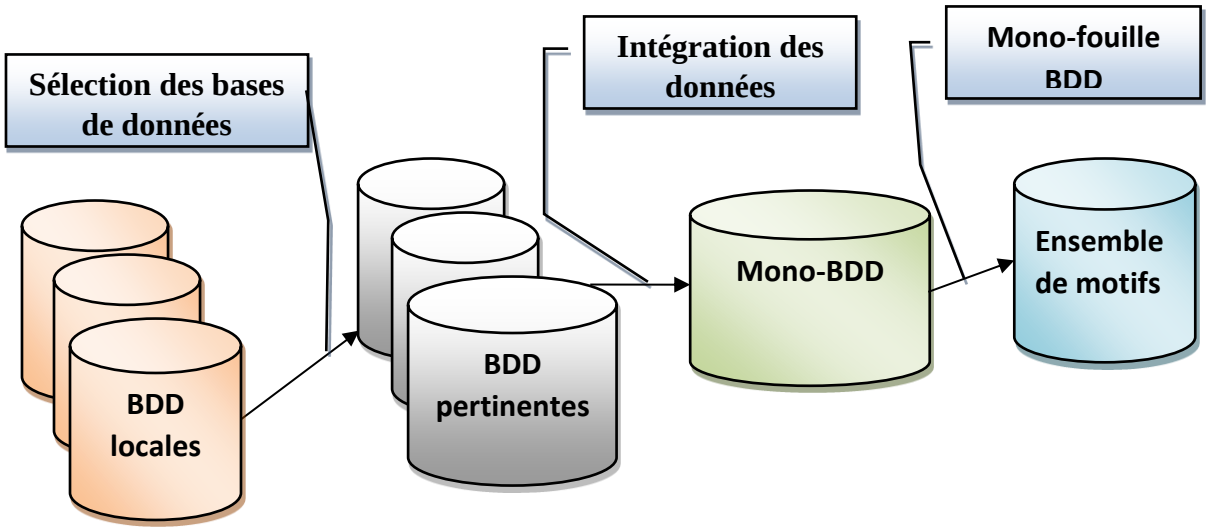



**FIGURE 1.5-** PROCESSUS TRADITIONNEL DE LA FOUILLE MULTI-BASES DE DONNEES AVEC SELECTION DES BDD PERTINENTES.

Pour faire face à la voluminosité des données, *la fouille de données distribuée* (DDM) peut être appliquée pour traiter les différentes possibilités de distribution et de traitement des données. Sans oublier le *méta-apprentissage hiérarchique* [Prodromidis and Stolfo 1998] qui a pour objectif de traiter efficacement les grandes quantités de données en combinant les prédictions des motifs apprises dans un environnement parallèle et distribué.

Cependant, pour traiter les problèmes liés à la confidentialité des données et l'identification des nouveaux types de motifs, [Zhang et al. 2003] ont proposé un nouveau procédé basé sur l'*analyse des motifs locaux* qui est décrit comme suit :

Etant donné *n* bases de données dans une grande société multi-branches, la fouille multi-bases de données est exécutée en trois étapes :

1. La recherche de la meilleure classification des multi-bases de données,
2. L'identification de nouveaux types de motifs et
3. La synthèse des motifs locaux par pondération.

Le processus de FMBD est décrit d'une manière simplifiée dans la figure 1.6.

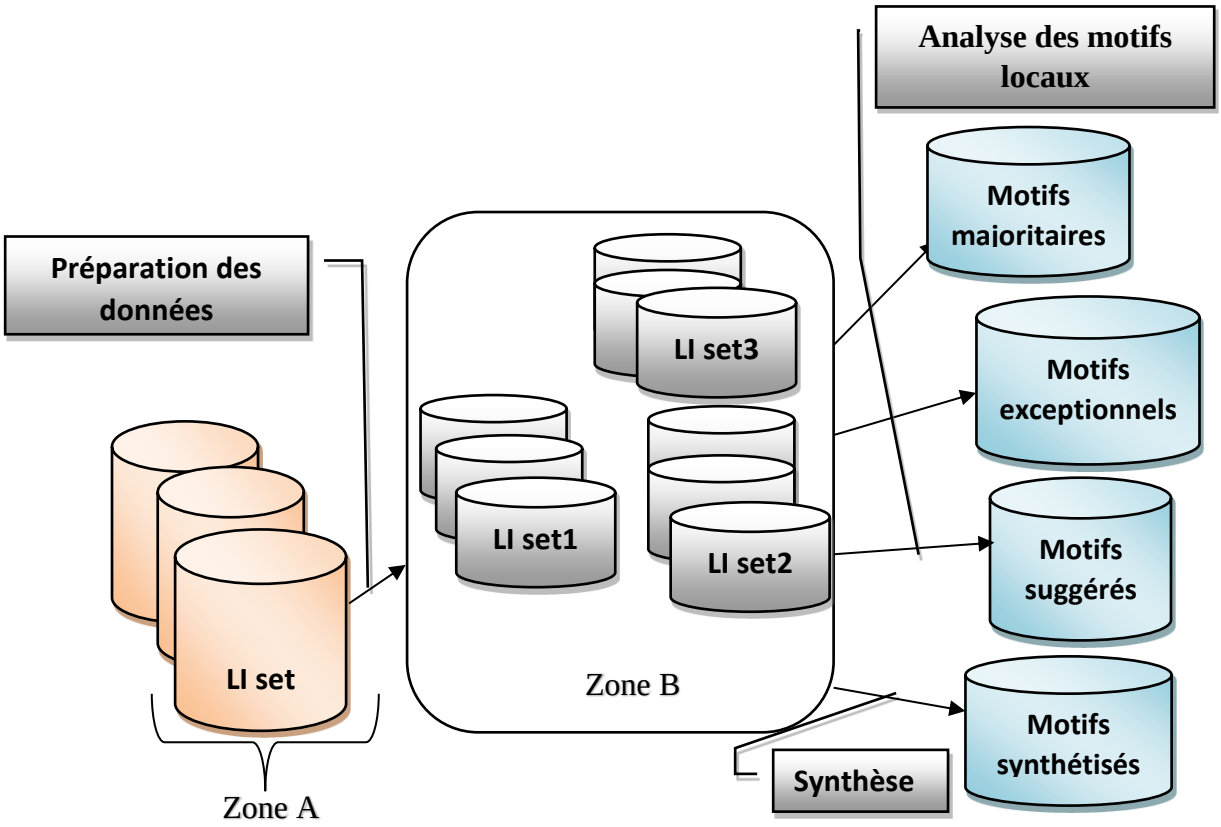



**FIGURE 1.6-** PROCESSUS DE FOUILLE MULTI-BASES DE DONNEES BASE SUR L'ANALYSE DES MOTIFS LOCAUX.

- Dans la figure 1.6**,** la zone «A» contient *n* ensembles de motifs locaux.
- La procédure « Préparation des données » permet le nettoyage des données et le clustering des *n* ensembles de motifs locaux dans des groupes similaires et distincts les uns des autres notés «LI set1», «LI set2» et «LI set3».
- Pour chaque cluster «LI set1», «LI set2» et «LI set3», la procédure «Analyse des motifs locaux » permet de rechercher de nouveaux types de motifs, comme les motifs *majoritaires*, les motifs *exceptionnels* et les motifs *suggérés.*
- La procédure «Synthèse» est utilisée pour synthétiser les motifs locaux de chaque groupe «LI set1», «LI set2» et «LI set3» de la zone « B» en motifs globaux.

## 3.3 Enjeux de la fouille multi-bases de données

La fouille multi-bases de données n'est pas toujours évidente à exécuter suite aux difficultés suivantes qu'on peut rencontrer en pratique :

### 3.3.1 Difficulté dans la préparation des données dans les multi-bases :

A cause des caractéristiques diverses des données dans les multi-bases, il est important de préparer et prétraiter les données avant de procéder à la fouille multi-bases de données. Parmi ces caractéristiques on peut citer :

- **Différences de nom :**
  Dans les multi-bases de données, le même attribut ou item peut posséder différents noms (des synonymes) dans différentes bases de données locales et le système multi-bases de données doit reconnaître l'équivalence sémantique des items et de faire correspondre les différents noms locaux à un nom global unique. Comme il peut y avoir aussi différents items de données qui ont le même nom (des homonymes) dans différentes bases de données locales, le système multi-bases de données doit reconnaître la différence sémantique entre les items et faire correspondre les noms communs à différents noms globaux.

- **Différences de format** :
  En particulier, les différences de type, de domaine, d'échelle et de précision des données. Par exemple, un numéro de matricule est représenté par un entier dans une base de données et par une chaîne alphanumérique dans une autre. Généralement les systèmes multi-bases de données résolvent les différences de format en définissant des fonctions de transformation entre les représentations locales et globales.

- **Différences structurelles** :
  Un item peut avoir une valeur unique (une clé) dans une base de données et plusieurs valeurs dans une autre. Un objet peut être représenté comme une relation unique dans un endroit ou par des relations multiples dans un autre.

Le même item de donnée peut être une valeur (réelle, entière, booléenne…) dans une table, un attribut dans une autre table et une relation dans une autre base de données.

- **Les données contradictoires** :
  Dans un système multi-bases de données, une perte d'information peut arriver à cause des mises à jour incomplètes (engendrant des valeurs manquantes), des erreurs ou des ressources insuffisantes pour maintenir ces données. Un problème plus grave se pose lorsque deux bases de données enregistrent le même item en lui attribuant des valeurs différentes. Les valeurs peuvent différer en raison d'une erreur ou en raison de la sémantique implicite des items.

Les données dans les multi-bases sont souvent différentes dans la structure et dans le contenu. Par conséquent, un prétraitement des données est souvent requis avant tout processus de fouille multi-bases.

### 3.3.2 Difficulté à rechercher de nouveaux types de motifs :

En utilisant les techniques traditionnelles de fouille multi-bases de données (e.i., la fouille de l'ensemble des bases intégrées) nous pouvons seulement identifier les motifs globaux de toute l'entreprise multi-branches. Ainsi, l'information qui reflète la distribution des motifs locaux sera perdue et l'identification des motifs *exceptionnels* deviendra impossible. En effet, pour la prise de décisions globales ou régionales, il est évident de voir le siège central de la société s'intéresser par les motifs découverts dans la plupart de ses branches (motifs majoritaires) ou bien par les motifs découverts dans certaines succursales (motifs exceptionnels).

En général, les motifs dans les multi-bases de données peuvent être divisés en 4 types :

- **Les motifs locaux :**
  Chaque succursale d'une société multi-branches possède ses fonctions individuelles, son propre plan de gestion et sa politique de développement. Afin d'identifier ses propres motifs locaux, chaque succursale devrait analyser uniquement les lignes de données brutes dans sa base de données locale. Ensuite, chaque succursale peut partager ses

motifs locaux avec d'autres succursales ou bien les transmettre au siège central de la société pour la prise des décisions globales.

- **Les motifs majoritaires :**
  Ce sont des motifs découverts par la plupart des succursales. Ils reflètent les caractéristiques communes aux succursales et sont généralement utilisés pour les prises des décisions globales.

- **Les motifs exceptionnels :**
  Ce sont des motifs pris en charge par seulement quelques succursales. Ils reflètent l'individualité de ces dernières et sont généralement utilisés pour créer des politiques spécifiques pour certaines succursales afin de prédire la vente des nouveaux produits par exemple.

- **Les motifs suggérés** :
  Ce sont des motifs qui ont peu nombre de voix mais ils peuvent aussi contribuer à améliorer la qualité des motifs globaux synthétisés.

### 3.3.3 Difficulté à traiter la voluminosité des données collectées :

Collecter toutes les données issues des multi-bases peut créer un énorme entrepôt de données qui va nécessiter une énorme capacité de stockage et un temps de calcul immense. Une meilleure approche pour faire face à la voluminosité des données est d'abord de classer les bases de données en groupes ou classes similaires. Ensuite les données d'une même classe peuvent être intégrées en une seule base pour la découverte de connaissances en utilisant les techniques de fouille existantes. La fouille parallèle est parfois utile pour faire face aux bases de données volumineuses.

### 3.3.4 Difficulté à maintenir la confidentialité des données :

Il est possible que certaines bases de données d'une organisation puissent accepter de partager leurs motifs, mais pas leurs lignes de données brutes. Cela est dû aux raisons suivantes :

- Certains types de données telles les données commerciales sont secrètes pour des raisons de concurrence et il n'est pas possible de les partager.
- Ré-analyser les lignes de données brutes est coûteux et partager les motifs ne l'est pas.
- Les décideurs inexpérimentés ne savent pas comment faire face à d'énormes quantités de données.

A partir des difficultés citées ci-dessus, de nouvelles stratégies de FMBD ont vu le jour.

### 3.4 Développement de stratégies de FMBD

Pour faire face aux difficultés rencontrées lors de l'application de la FMBD traditionnelle**,** le développement de stratégies de préparation des données et d'identification des nouveaux types de motifs est indispensable. Ces stratégies sont décrites comme suit :

#### 3.4.1 La préparation des données :

En général, la préparation des données prend plus de temps et nécessite plus de défis à relever. L'importance de la préparation des données peut être illustrée comme suit :

- Les données du monde réel sont bruitées et impures,
- Les systèmes de fouille performants exigent des données de haute qualité et
- La nécessité de synthétiser des motifs de haute qualité.

Par conséquent, le développement des méthodes de préparation des données est à la fois difficile et primordial. Il y a quatre problèmes clés dans la préparation des données:

- Développement des techniques de nettoyage des données,
- Développement d'un système logique d'identification des connaissances de qualité dans les différentes sources de données,
- Développement d'un système logique pour résoudre les conflits et incohérences des connaissances dans les différentes sources de données et
- Conception d'un clustering de base de données indépendant de toute application.

**a) Techniques de nettoyage des données :**

Les techniques de nettoyage des données ont été largement étudiées et appliquées dans la reconnaissance des formes, l'apprentissage machine et le datamining. Pour la fouille multi-bases de données, le nettoyage des données distribuées relève plus de défis que le nettoyage traditionnel des données dans une seule base. Cela est dû aux données qui peuvent entrer en conflit au sein des multi-bases. Les techniques suivantes sont utilisées pour générer des données de haute qualité pour la fouille multi-bases de données :

- Remplissage des valeurs manquantes et suppression de l'ambiguïté lors de la récupération des données incomplètes.
- Vérification de la cohérence des noms et formats de données, correction des erreurs et suppression des valeurs aberrantes.
- Résolution des conflits des données par l'intervention de l'expert du domaine pour régler les divergences.

**b) Identification des connaissances dans les multi-bases :**

Les données issues des sites-web, appelées sources de données externes, contiennent souvent du bruit. Par conséquent, il est nécessaire de développer un système logique d'identification des connaissances de haute qualité basé sur les propriétés suivantes:

- La crédibilité des données : l'information détenue par la donnée doit être vraie.
- La cohérence des données : Les connaissances extraites d'une source de données doivent être non-contradictoires.

**c) Conception d'un clustering de bases de données indépendant de toute application** :

En pratique, la sélection de base de données développée par [Liu et al. 2001] [Liu et al. 1998] est exécutée afin d'identifier les bases de données pertinentes pour chaque application donnée. Cependant, les utilisateurs peuvent vouloir explorer les multi-bases de données sans spécifier aucune requête. Par conséquent, l'approche de sélection des bases de données ne va pas fonctionner dans ce cas.

Prenons comme exemple une grande société à 25 succursales distribuées comme suit :

- 5 succursales enregistrant un maximum de vente des produits alimentaires,
- 7 succursales enregistrant un maximum de vente des produits vestimentaires et
- 13 succursales enregistrant un maximum de vente des produits cosmétiques.

Si on assume que chaque succursale possède au moins une base de données, alors il devient important de classer ces multi-bases de données en trois groupes (en fonction des items les plus demandés et les plus achetés) avant qu'elles ne soient explorées. Cette approche est dite indépendante de l'application utilisateur et sera discutée en détail dans les prochaines sections.

Pour effectuer une classification de base de données efficace et indépendante de toute application, il est nécessaire de :

- Concevoir des métriques pour déterminer la similarité entre bases de données,
- Concevoir des mesures pour déterminer la meilleure classification et
- Concevoir des algorithmes efficaces en matière de temps d'exécution et d'espace mémoire.

### 3.4.2 Identification de nouveaux types de motifs dans les multi-bases de données :

Le développement de techniques pour la recherche de nouveaux types de motifs comprend :

- La conception d'une stratégie efficace d'analyse des motifs locaux,
- L'identification des motifs majoritaires,
- La recherche des motifs exceptionnels et
- La synthèse des motifs globaux.

#### a) Conception d'une stratégie d'analyse des motifs locaux :

La stratégie d'analyse des motifs locaux est efficace pour la découverte de nouveaux types de motifs dans les multi-bases. Cependant, l'ensemble de tous les motifs locaux peut toutefois être très grand. Ainsi, cette stratégie peut devenir difficile à réaliser avec un traitement centralisé.

En outre, il n'est pas toujours évident d'identifier lequel de ces motifs locaux est vraiment utile. D'où la nécessité d'améliorer cette stratégie pour filtrer efficacement l'ensemble des motifs locaux intéressants à toute l'entreprise multi-branches.

**b) Identification des motifs majoritaires :**

Ces motifs sont importants pour la prise de décision globale au niveau de toute l'entreprise et les techniques de fouille traditionnelles ne permettent pas de les identifier. D'où la nécessité de concevoir une stratégie de fouille de *motifs majoritaires* basée sur l'analyse des motifs locaux.

**c) Découverte des motifs exceptionnels :**

Tandis que les *motifs majoritaires* sont utiles pour la prise de décisions globales, les décideurs sont également intéressés par les motifs exceptionnels lorsque des décisions spéciales doivent être prises pour cibler certaines branches. Cela permettrait de comprendre les caractéristiques communes à certaines branches et de répondre à des questions du genre : Quels sont les produits les plus vendus dans les succursales situées dans les villes côtières.

**d) La synthèse des motifs globaux :**

Les succursales d'une entreprise multi-branches peuvent avoir différents poids d'importance et ainsi influencer différemment la prise de décision globale. Par exemple, si les ventes de la succursale « A » sont 4 fois plus que celles de la succursale « B », alors la succursale « A » est considérée plus importante que la succursale « B ». Par conséquent, les décisions de l'entreprise peuvent être grandement influencées par les succursales ayant le plus de ventes. En outre, les motifs locaux peuvent avoir différents supports dans les différentes succursales.
D'où le besoin de concevoir un modèle pondéré pour la synthèse des motifs globaux prenant en considération les poids et les supports des motifs locaux dans les multi-bases.

## 4 La Fouille multi-bases de données et le distribué

La découverte des motifs locaux au niveau des multi-bases de données peut être réalisée en utilisant des techniques traditionnelles de fouille de données. Cependant, si une base de données est très volumineuse, cela nécessiterait de grandes capacités de stockage et de traitement centralisées, imposant ainsi le recours aux algorithmes parallèles et distribués.

En effet, la fouille parallèle des règles d'association MARP (en anglais Mining Association Rule Upon Paralleling) [Agrawal and Shafer 1996] [Park et al. 1995] [Parthasarathy et al. 2001] emploie plusieurs machines pour l'implémentation et l'exécution parallèle des algorithmes de datamining. Bien que ces algorithmes soient efficaces, ils nécessitent plus de ressources de calcul telles que les machines et les logiciels de gestion des modules parallèles.

Néanmoins, la disponibilité de plus en plus des grilles informatiques, fournit des ressources puissantes et à faible coût pour des traitements de calcul distribués de haute performance. En utilisant ces ressources, le datamining distribué va permettre de partitionner horizontalement ou verticalement une base de données volumineuse à travers les nœuds d'une grille. Ces partitions sont ensuite fouillées d'une manière parallèle afin de réduire la complexité du traitement et diminuer l'espace de recherche à explorer.

Contrairement à la fouille multi-bases de données, les problèmes d'hétérogénéité et d'incohérence des données n'existent pas dans l'application de la fouille de données distribuées. Car il s'agit d'explorer les fragments d'une même base de données à travers les différents nœuds de la grille.

## 5 Applications de la fouille multi-bases de données

Le domaine de la fouille multi-bases de données est en pleine expansion et ses champs d'application dépendent de la complexité des problèmes rencontrés et qui sont susceptibles d'augmenter dans le futur.

Dans le domaine commercial, c'est *la fouille multi-bases de données transactionnelle* qui est souvent appliquée vu l'existence des entreprises opérantes à partir de plusieurs succursales localisées dans différentes régions géographiques. Certaines de ces succursales sont totalement opérationnelles et collectent des données transactionnelles de façon continue dans leurs bases de données locales. Donc pour ces entreprises, la prise de

décisions globales sera plus influencée par l'analyse des données transactionnelles distribuées à travers leurs branches.

Dans le domaine bio-informatique, c'est *la fouille multi-relationnelle* qui est appliquée vu la diversité des données qui requièrent un schéma relationnel complexe. Dans un tel schéma, on a besoin par exemple d'encoder les séquences d'ADN et les localisations des gènes ainsi que les relations biochimiques diverses entre les protéines. Donc, l'analyse des enregistrements détaillés des phénotypes humains encodés dans ces tables multi-relationnelles va permettre la découverte des relations entre l'expression des gènes et la présence ou la progression d'une maladie.

Bien que tous les domaines d'application de la fouille multi-bases de données soient intéressants et importants, nous nous sommes intéressés à *la fouille multi-bases de données transactionnelle* vu son importance à résoudre les problèmes décisionnels rencontrés dans les organisations multi-branches.

## 6 Conclusion

Nous avons présenté dans ce chapitre le processus traditionnel de la fouille multi-bases de données et ses limites. Nous avons aussi démontré que le clustering des multi-bases de données constitue un moyen efficace pour réduire le coût de recherche des motifs utiles dans les multi-bases et améliorer leur qualité.

Dans le deuxième chapitre, nous allons étudier l'intérêt de concevoir des algorithmes efficaces pour la recherche d'une meilleure classification des multi-bases de données. Nous présentons aussi le principe des approches de clustering existantes. Cette étude est conclue par une synthèse et une étude critique des différentes approches étudiées.

# Chapitre 2 -

# Approches et algorithmes de clustering des multi-bases de données

# 1 Introduction

Une stratégie de classification des multi-bases de données peut être soit *dépendante* ou *indépendante* de l'application utilisateur. Par définition, une classification dépendante de l'application utilisateur, appelée aussi *sélection des bases de données pertinentes*, permet d'identifier les bases de données les plus pertinentes à une application ou une requête utilisateur en utilisant le facteur de pertinence *RF* proposé par [Liu et al. 2001] [Liu et al. 1998]. Ce facteur permet d'évaluer la déviation entre la requête utilisateur *Q* et les attributs des différentes bases de données. La figure 2.1 illustre les étapes du processus de sélection des bases de données pertinentes proposée par [Liu et al. 2001] [Liu et al. 1998].

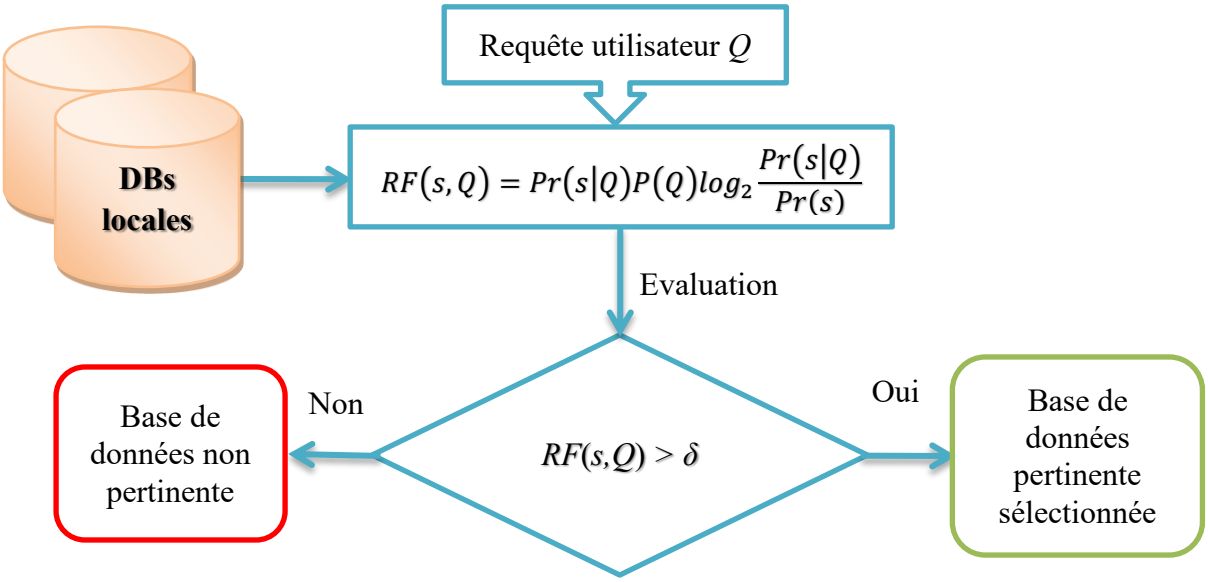



**FIGURE 2.1 -** LE PROCESSUS DE SELECTION DES BASES DE DONNEES PERTINENTES.

Une fois identifiées et sélectionnées, les bases de données pertinentes peuvent ensuite être fouillées pour la découverte de motifs utiles en relation avec la requête utilisateur. Cette stratégie est efficace pour la réduction des coûts de recherche pour les applications de fouille de données. La *sélection des bases de données* doit être exécutée plusieurs fois dans l'ordre d'identifier les bases de données pertinentes pour deux ou plusieurs applications du monde réel. Cependant, lorsqu'une tâche de fouille de données est sans référence à aucune application spécifique, les techniques dépendante-application ne sont pas suffisantes.

De l'autre côté, une stratégie de classification indépendante de l'application utilisateur fait référence au processus de clustering de plusieurs bases de données relatives aux différentes succursales d'une entreprise multi-branches. Ce processus utilise une mesure de similarité inter-bases de données pour déterminer la pertinence entre les bases et cela indépendamment de toute application. En outre, une mesure d'évaluation de la qualité du clustering est utilisée aussi afin de déterminer la meilleure classification des bases de données. Cette stratégie est appropriée pour des applications de fouille à usage général et universel. En effet, une fois le processus de clustering achevé, chaque cluster de bases de données peut être fouillé pour répondre à une requête utilisateur donnée.

Quelques approches de clustering multi-bases de données ont été conçues dans la littérature [Adhikari et al. 2010a] [Li et al. 2009] [Liu et al. 2013] [Wu et al. 2005]. La figure 2.2 illustre les étapes générales de ce processus.

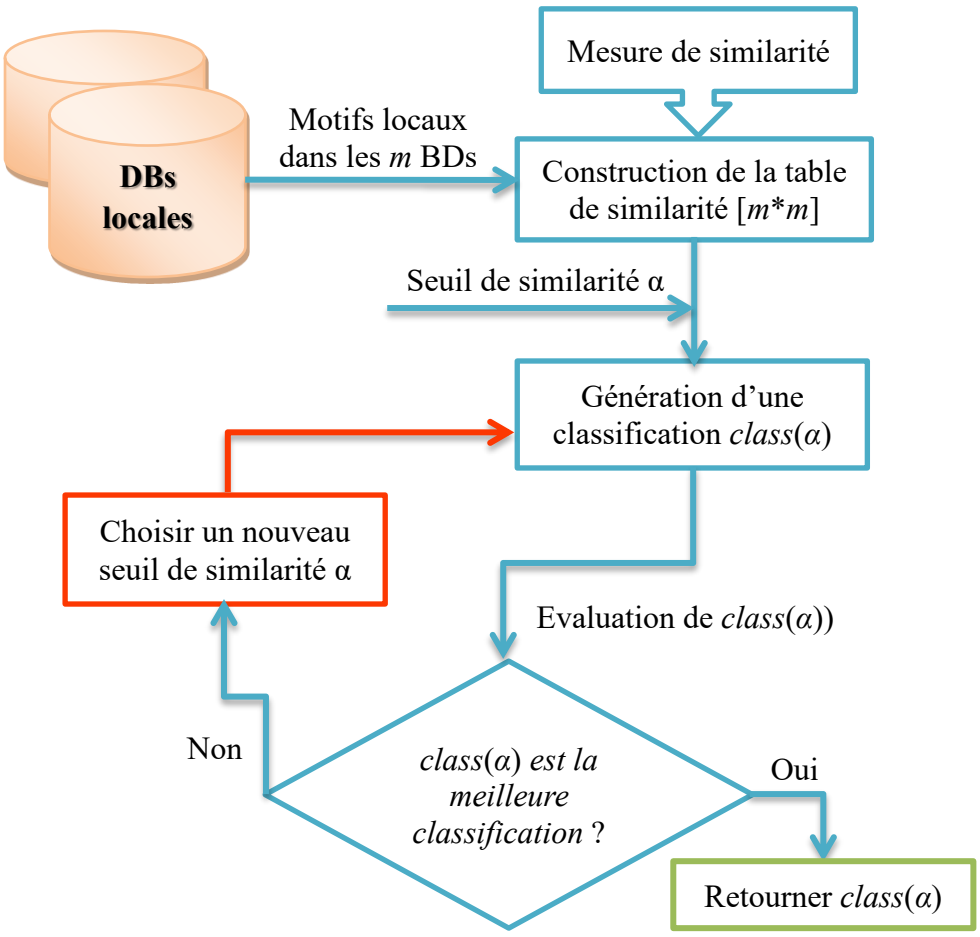



**FIGURE 2.2-** LE PROCESSUS DE CLUSTERING DES MULTI-BASES DE DONNEES

Les approches de clustering des multi-bases diffèrent dans les points suivants :

- La mesure de similarité utilisée pour déterminer la pertinence entre bases de données (Par exemple : mesure basée sur les items, itemsets fréquent ou items dans les règles d'association).
- La mesure utilisée pour déterminer la meilleure classification (Par exemple : minimiser la mesure basée sur la distance intra-classe ou bien maximiser la mesure basée sur la similarité intra-classe et la distance inter-classe).
- Les algorithmes conçus pour la génération et la recherche de la meilleure classification des bases de données.

Une approche de clustering de base de données efficace dépend de l'utilisation d'une mesure de similarité appropriée. En effet pour classer les multi-bases de données, nous avons besoin de savoir ce qui est utilisé pour évaluer la pertinence entre les différentes bases de données. En outre, un clustering des multi-bases efficace dépend aussi de l'algorithme utilisé pour la génération et la recherche de la meilleure classification. Ces algorithmes doivent trouver la meilleure classification en utilisant le moins de ressources de calcul et de mémoire possible.

Les algorithmes de clustering traditionnel conçus dans la littérature tels que *BIRCH* [Zhang et al. 1997] and *K-MEANS* [Na et al. 2010] sont basés sur des attributs métriques**.** Un attribut métrique est un attribut dont les valeurs peuvent être représentées par des coordonnées explicites dans un espace euclidien. Ainsi les techniques de clustering traditionnel, peuvent ne pas fonctionner, puisque nous nous intéressons au clustering des multi-bases de données.

Notre étude sera restreinte uniquement sur le clustering des multi-bases de données transactionnelles. Dans ce chapitre, nous allons présenter quelques approches de clustering des multi-bases de données ainsi que leurs algorithmes tout en étudiant leurs performances. Ce chapitre sera conclu par une synthèse et une étude critique des différentes approches étudiées.

## 2 Complexité et performance d'un algorithme

Dans cette section, nous allons rappeler l'analyse de la complexité des algorithmes en général et aussi les critères utilisés pour évaluer la performance des algorithmes de clustering des multi-bases de données.

### 2.1 Analyse théorique de la performance d'un algorithme

En général, l'analyse de la performance d'un algorithme est la détermination de la quantité de l'espace mémoire, le temps d'exécution et les autres ressources nécessaires au déroulement de cet algorithme. Cette analyse inclut aussi la détermination d'une fonction qui établit un rapport entre la taille de l'entrée (input) de l'algorithme avec le nombre d'étapes qu'il prend (*complexité temporelle*) ou bien avec le nombre d'unités mémoires qu'il utilise (*complexité spatiale*). En général, un algorithme est dit performant, lorsque les valeurs de cette fonction sont petites. Puisque l'algorithme peut avoir des comportements différents pour des entrées de la même taille, la fonction décrivant la performance est en fait une borne supérieure de la performance réelle de l'algorithme, qui est déterminée à partir des entrées de l'algorithme choisies au pire des cas. L'analyse de la performance des algorithmes est une partie importante de la théorie de la complexité, qui fournit des estimations théoriques des ressources requises par un algorithme pour résoudre un problème de calcul donné. Ces estimations fournissent un aperçu des directions à prendre pour concevoir des algorithmes performants.

Dans l'analyse théorique des algorithmes, les complexités temporelles et spatiales sont estimées dans un sens asymptotique pour une entrée arbitraire en utilisant les notations O, Ω et Θ. Par exemple, la recherche dichotomique dans une liste triée de $n$ éléments est réalisée en un nombre d'étapes proportionnel au logarithme de la longueur de la liste ou bien en un temps logarithmique noté $O(log_2(n))$. Ainsi, si on exécute la recherche dichotomique sur une liste triée de taille $n$ et si on peut garantir que chaque recherche d'un élément dans cette liste peut être réalisée en une unité de temps, alors l'algorithme a besoin de $log_2(n+1)$ unités de temps au plus pour trouver un élément donné. Les estimations asymptotiques sont utilisées car différentes implémentations du même algorithme peuvent différer en matière d'efficacité.

## 2.2 Benchmarking et analyse empirique

L'analyse du temps d'exécution d'un algorithme permet d'estimer la croissance de son temps d'exécution en fonction de la croissance de la taille de l'entrée *n*. En effet, le temps d'exécution est d'une grande importance en informatique car un code peut prendre des secondes, des heures, voire des années pour finir son exécution, tout dépend de l'algorithme. Puisque les algorithmes sont indépendants de toute plateforme (i.e., un algorithme donné peut être implémenté avec langage de programmation arbitraire sur un PC quelconque qui fonctionne avec un système d'exploitation quelconque aussi), il n'est pas suffisant d'utiliser une approche empirique pour mesurer la performance comparative d'un ensemble d'algorithme donné.

**Exemple :**

Si on prend un programme qui recherche une entrée spécifique dans une liste ordonnée de taille *n*. Supposons que ce programme a été implémenté sur un ordinateur performant « A » en utilisant un algorithme de recherche linéaire. De l'autre côté, ce programme a été implémenté sur un ordinateur plus lent « B » en utilisant un algorithme de recherche dichotomique. En analysant le temps d'exécution des deux programmes, on peut constater que lorsque les valeurs de la taille de l'entrée *n* sont très petites, l'algorithme implémenté sur « A » est beaucoup plus rapide que celui implémenté sur « B ». Cependant, dès que la valeur *n* dépasse un certain seuil, le programme exécuté sur « A » va devenir plus lent que celui exécuté sur « B ». Ceci est dû au fait que l'ordinateur « A » exécute un code exposant un taux de croissance linéaire du temps d'exécution (en fonction de la taille de l'entrée *n*). De l'autre côté, l'ordinateur « B », exécute un code exposant un taux de croissance logarithmique du temps d'exécution. Donc, même si l'ordinateur « A » est beaucoup plus rapide, l'ordinateur « B » va surpasser l'ordinateur « A » en matière de temps d'exécution lorsque la taille de l'entrée *n* va atteindre un certain seuil. D'où l'importance de l'analyse de la complexité temporelle et spatiale d'un algorithme.

L'approche empirique utilisée pour déterminer si un programme « A » est plus rapide qu'un programme « B » est d'écrire un petit programme appelé *micro-benchmark*. En effet, écrire un *micro-benchmark* pour un code Java est beaucoup plus difficile que de le faire pour un code C++. Cela revient au fait qu'il est compliqué d'écrire et d'interpréter des benchmarks pour des langages de programmation dynamiquement compilés, dont la phase

de compilation génère un byte-code portable indépendant de tout système d'exploitation qui sera interprété en code natif durant l'exécution. Donc, pour avoir des résultats plus au moins justes, nous devons exécuter les codes plus d'une fois pour permettre à la JVM (machine virtuelle java) de s'échauffer. Cela signifie d'exécuter le code durant un temps assez suffisant pour permettre à la JVM de faire les optimisations de code nécessaires telles que le remplacement de l'appel d'une méthode par le corps de la méthode, élimination des sous-expressions communes, déroulement des boucles, élimination des codes morts, etc. De plus, lorsqu'un code JAVA s'exécute plusieurs fois, il est remplacé par le code machine qui peut être exécuté directement sur la plateforme cible.

## 2.3 Critères de performance du clustering des multi-bases de données

Un algorithme de clustering pour la fouille multi-bases de données est dit *performant* si *indépendamment* de toute application, il peut trouver la *meilleure* classification des *n* multi-bases de données en utilisant le moins de ressources possibles en matière de *temps d'exécution* et d'*espace mémoire*.

Une classification est dite *meilleure* si elle satisfait les conditions suivantes :

- Elle est générée en utilisant le moins de ressources (temps d'exécution et espace mémoire) possible.
- Possède la plus grande similarité inta-classe : les bases de données qui se trouvent dans la même classe sont les plus similaires possibles partageant le plus grand nombre d'itemsets fréquents avec des valeurs de support très proches.
- Possède la plus grande distance inter-classe : les bases de données qui se trouvent dans des classes différentes sont les plus dissimilaires possibles partageant le moins nombre d'itemsets fréquents avec des valeurs de support très différentes.
- Le nombre de cluster dans une classification est différent de 1 et de *n*.
- L'analyse des itemsets fréquents dans les bases de données de chaque cluster va permettre de synthétiser un grand nombre d'itemsets globaux dont les supports doivent être proches de ceux obtenus par la mono-fouille de données (le cas où les bases de données de chaque cluster sont intégrées en un seul ensemble de données pour la découverte des itemsets fréquents globaux).

# 3 Algorithmes de classification des multi-bases de données dans la littérature

Quelques approches de classification des multi-bases de données ont été proposées dans la littérature. Dans les paragraphes suivants, nous présentons ces différentes approches ainsi que leurs algorithmes de clustering.

## 3.1 Identification des bases de données pertinentes [Liu et al. 2001] [Liu et al. 1998]

Selon les auteurs de cette approche, la première phase à réaliser dans le processus de la fouille multi-bases de données est d'identifier les bases les plus pertinentes à une application utilisateur. En l'absence de cette phase, le processus de fouille de donnée devient lent et inefficace. Comme dans chaque processus de découverte de connaissance, la sélection des bases de données pertinentes permet de :

1. Eviter la jointure forcée des bases de données pertinentes avec celles non pertinentes,
2. Réduire l'espace de recherche à explorer,
3. Augmenter l'importance statistique des connaissances découvertes et
4. Trouver des motifs intéressants et utiles.

C'est pourquoi une mesure de pertinence a été proposée pour découvrir des relations au sein des attributs des multi-bases. Un algorithme d'identification des bases de données pertinentes a été conçu et une série d'expérimentations a été menée afin de vérifier la performance de la mesure proposée.

### 3.1.1 Méthode proposée :

Le problème d'indentification des bases de données pertinentes peut être défini comme suit. Soient $n$ tables relationnelles, $D_k (1 \leq k \leq n)$, chacune ayant un certain nombre d'attributs. Soit $Q$ une requête utilisateur exprimée sous la forme de « $A_i$ *relop* $C$ » où $A_i$ est un attribut, *relop* est un opérateur relationnel ($\leq, \geq, \neq, =, <, >$) et $C$ est une valeur dans le domaine de $A_i$. Alors, l'indentification des bases de données pertinentes revient à sélectionner les tables pertinentes contenant des informations fiables, spécifiques et d'une importance statistique par rapport à la requête $Q$. Ces informations doivent être cohérentes et justes avec un certain degré de précision au-dessus du seuil défini par l'utilisateur.

**a) Mesure de pertinence :**

Afin d'identifier les bases de données pertinentes à une tâche de fouille de données, les auteurs de l'approche ont établi la mesure de pertinence définie comme suit. Soit *« A relop C »* un sélecteur noté *s*, où *A* est le nom d'un attribut non référencé dans la requête *Q*. Alors le facteur de pertinence *RF* d'un sélecteur *s* est défini comme suit :

$$RF(s,Q) = Pr(s|Q)P(Q)log_2 \frac{Pr(s|Q)}{Pr(s)}$$

Pr(*Q*) et Pr(*s*) sont des probabilités estimées par le rapport du nombre d'enregistrements vérifiant *s* et *Q* sur le nombre de tous les enregistrements. Pr(*s*|*Q*) représente la fréquence d'apparition de *s* quand la requête *Q* est satisfaite. La logique de la définition d'une telle mesure revient au fait que le rapport $\frac{\Pr(s|Q)}{\Pr(s)}$ représente le degré de déviation entre Pr(*s*|*Q*) et Pr(*s*). Afin de trouver les bases de données pertinentes, il faudrait s'intéresser au cas où $\frac{\Pr(s|Q)}{\Pr(s)}$ est plus grand que 1 car cela indique que *s* apparait plus souvent avec *Q* qu'en étant seul, donc *s* et *Q* sont corrélés. Un autre poids Pr(*Q*) est introduit pour tenir compte de la fréquence d'apparition de *Q*. En effet, si Pr(*Q*) est proche de 0, alors *Q* apparait rarement dans la base de données. Ainsi on ne peut pas conclure statiquement l'existence d'une information reliée à *Q*.

Donc, plus la valeur de *RF* est grande, plus le sélecteur *s* est pertinent vis-à-vis de la requête *Q*. Si la valeur de *RF* de tous les sélecteurs de base de données est proche ou inférieur de 0, alors la base de données est non pertinente par rapport à *Q*.

Ainsi, selon la définition du facteur *RF*, la pertinence d'une base de données (ou table) peut être définie comme suit. Un sélecteur *s* est pertinent par rapport à *Q* si *RF*(*s*,*Q*) > δ où δ est un seuil défini par l'utilisateur et qui est supérieur à 0. Une table est dite pertinente par rapport à *Q* s'il existe au moins un sélecteur $s_j$, « $A_i$ relop *C* » tel que $s_j$ est pertinent par rapport à *Q*.

### 3.1.2 Complexité de l'algorithme [Liu et al. 2001] [Liu et al. 1998] :

Pour calculer *RF(s,Q)*, nous avons besoin de calculer les trois valeurs, *Pr(Q), Pr(s), Pr(Q∧ s).* Pour déterminer si une base de données est pertinente par rapport *Q*, nous avons besoin aussi de tester tous les sélecteurs, en scannant une seule fois la base de données. Supposant qu'il existe *n+1* attributs $A_0,\ldots,A_n$ dans la table *D* et l'attribut $A_0$ est référencé par *Q*. Pour chacun des autres attributs $A_i(1 \leq i \leq n)$, une table $S_i$ lui est associée afin de stocker les valeurs reliées aux sélecteurs de $A_i$. Une entrée dans la table $S_i$ est un triplet *(S_value, S_counter, SQ_counter). S_value* est une valeur du sélecteur (valeur unique de l'attribut $A_i$), *S_counter* est le nombre d'enregistrements vérifiant $A_i$=*S_value*. *SQ_counter* est le nombre d'enregistrements vérifiant *Q* et $A_i$=*S_value* en même temps. L'algorithme est illustré dans la figure 2.3.

L'algorithme proposé scanne la table *D* pour trouver tous les sélecteurs possibles et les insérer dans les tables $S_i$. Après le parcours de *D*, une fonction *ComputeRF* est appelée pour chaque sélecteur afin de calculer *RF*. S'il existe au moins une valeur *RF* supérieure au seuil prédéfini par l'utilisateur, l'algorithme s'arrête et retourne *TRUE* sinon l'algorithme retourne *FALSE* et dans ce cas la base est non pertinente. L'algorithme *RelevantDB* est composé de deux parties : la lecture de chaque enregistrement dans la table *D* et la recherche de l'entrée des sélecteurs pour mettre à jour les compteurs de chaque attribut dont la valeur a été trouvée dans l'enregistrement. Le coût de la première partie est proportionnel au nombre d'enregistrements dans *D*. Tandis que dans la deuxième partie, avec une structure de données de type dictionnaire et en utilisant une fonction de hachage, la recherche des entrée des sélecteurs va être constante, sans tenir compte du nombre de sélecteurs d'un attribut.

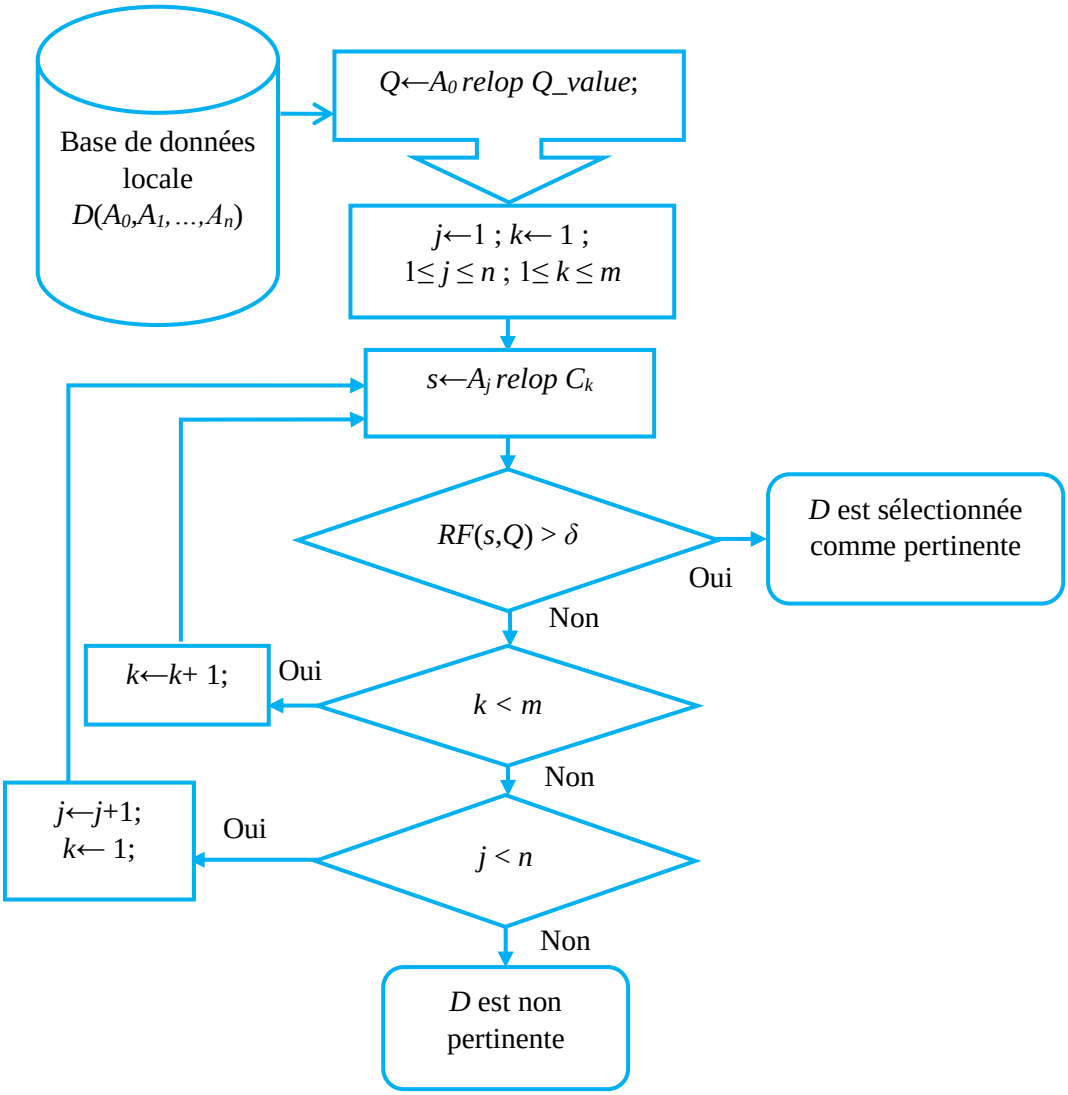



**FIGURE 2.3 –** ALGORITHME D'IDENTIFICATION DES BASES DE DONNEES PERTINENTES [LIU ET AL. 2001] [LIU ET AL. 1998].

Donc, la complexité temporelle de l'algorithme est $O(mNM)$, où $N$ et $M$ sont respectivement le nombre d'enregistrements moyen et le nombre d'attributs moyen dans les $m$ multi-bases de données.

## 3.2 La classification pour la fouille multi-bases de données [Wu et al. 2005]

La sélection des bases de données proposée par [Liu et al. 2001] [Liu et al. 1998] identifie les bases de données les plus pertinentes à une application réduisant ainsi le coût de recherche dans les multi-bases. Cependant, cette sélection des bases de données est dépendante de l'application de l'utilisateur. Ainsi, lorsque la fouille de donnée doit être

exécutée sans spécifier aucune requête utilisateur, la sélection des bases de données ne peut être appliquée.

Cela a motivé les auteurs [Wu et al. 2005] à concevoir une stratégie efficace pour une classification des multi-bases de données indépendamment de toute application utilisateur. Cette méthode de classification utilise une mesure de similarité afin regrouper les bases de données les plus similaires au sein d'une même classe. La valeur de similarité entre deux bases de données d'une même classe, doit être supérieure ou égale à un certain seuil. En faisant varier ce seuil, différentes classifications peuvent être formées. D'où la nécessité de chercher et d'identifier la meilleure classification parmi toutes les classifications trouvées. Pour cela, une autre mesure d'évaluation de la qualité d'une classification est utilisée. Ce type de classification est aussi connu sous le nom du *clustering* ou *classification non supervisée.*

Une fois le processus de clustering est achevé, chaque cluster de bases de données peut être fouillé pour trouver des motifs utiles à la prise de décision au niveau de chaque cluster de succursale. Pour illustrer l'intérêt de concevoir une telle approche, les auteurs [Wu et al. 2005] ont présenté l'exemple suivant. Considérons six bases de données D1, D2,…, D6 relatives aux six succursales d'une entreprise multi-branches comme suit :

$D_1$ = {(A,B,C,D) ;(B,C) ;(A,B,C) ;(A,C)}
$D_2$ = {(A,B) ;(A,C) ;(A,B,C) ;(B,C);(A,B,D)}
$D_3$ = {(B,C,D) ;(A,B,C) ;(B,C) ;(A,D)}
$D_4$= {(F,G,H,I,J) ;(E,F,H) ;(F,H)}
$D_5$ = {(E,F,H,J) ;(F,H) ;(F,H,J) ;(E,J)}
$D_6$= {(F,H,I,J) ;(E,H,J) ;(E,F,H) ;(E,I)}

Chaque base de données possède quelques transactions séparées par des points-virgules (;) et chaque transaction contient plusieurs items séparés par une virgule (,). En mettant ensemble les six bases de données dans TD=D1 ∪D2∪…∪D6 (24 transactions), nous pouvons remarquer qu'avec un support de seuil minimum *min.supp=0.5* ; il n'y a aucun itemset fréquent à découvrir à partir de TD comme le montre la figure 2.4.

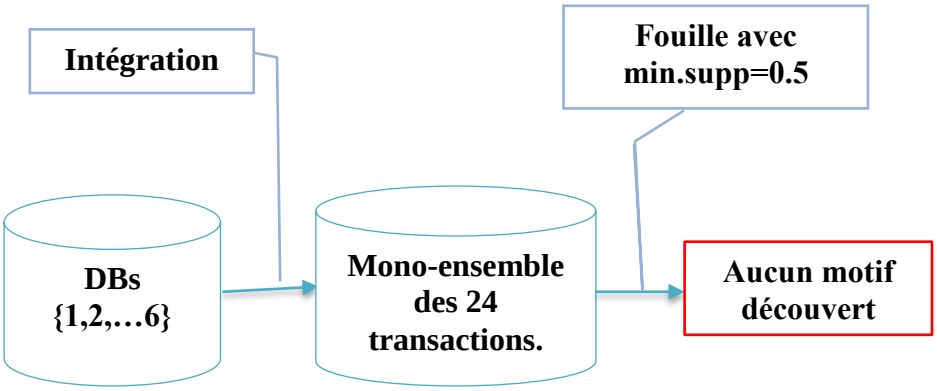



**FIGURE 2.4 –** APPLICATION DE LA FOUILLE MONOBASE DE DONNEES SUR L'ENSEMBLE INTEGRE DES 6 BASES DE DONNEES.

Procédons maintenant à la classification des six bases de données. En observant les transactions de chaque base de données, Il est évident que les six bases de données appartiennent à deux classes différentes. La première classe est pertinente aux items A, B, C et D. La seconde classe est pertinente aux items E, F, G, H, I et J.

Donc après le classement des six bases de données en deux groupes TD1={D1,D2,D3} (avec 13 transactions) et TD2={D4,D5,D6} (avec 11 transactions), une fouille de données a été appliquée sur l'union des transactions de chaque groupe individuellement et les résultats suivants ont été obtenus :

Pour *minsupp*=0.5, A, B, C, et BC sont des itemsets fréquents extraits à partir de TD1. Les itemsets fréquents E, F, H, J et FH sont extraits à partir de TD2 comme illustré dans la figure 2.5. A partir de ces observations, nous pouvons constater que la jointure forcée des multi-bases de données peut masquer les motifs utiles à cause des quantités énormes des données non pertinentes incluses dans le processus de fouille de données. D'où la nécessité de concevoir une approche de classification efficace pour améliorer la qualité des motifs découverts dans les multi-bases.

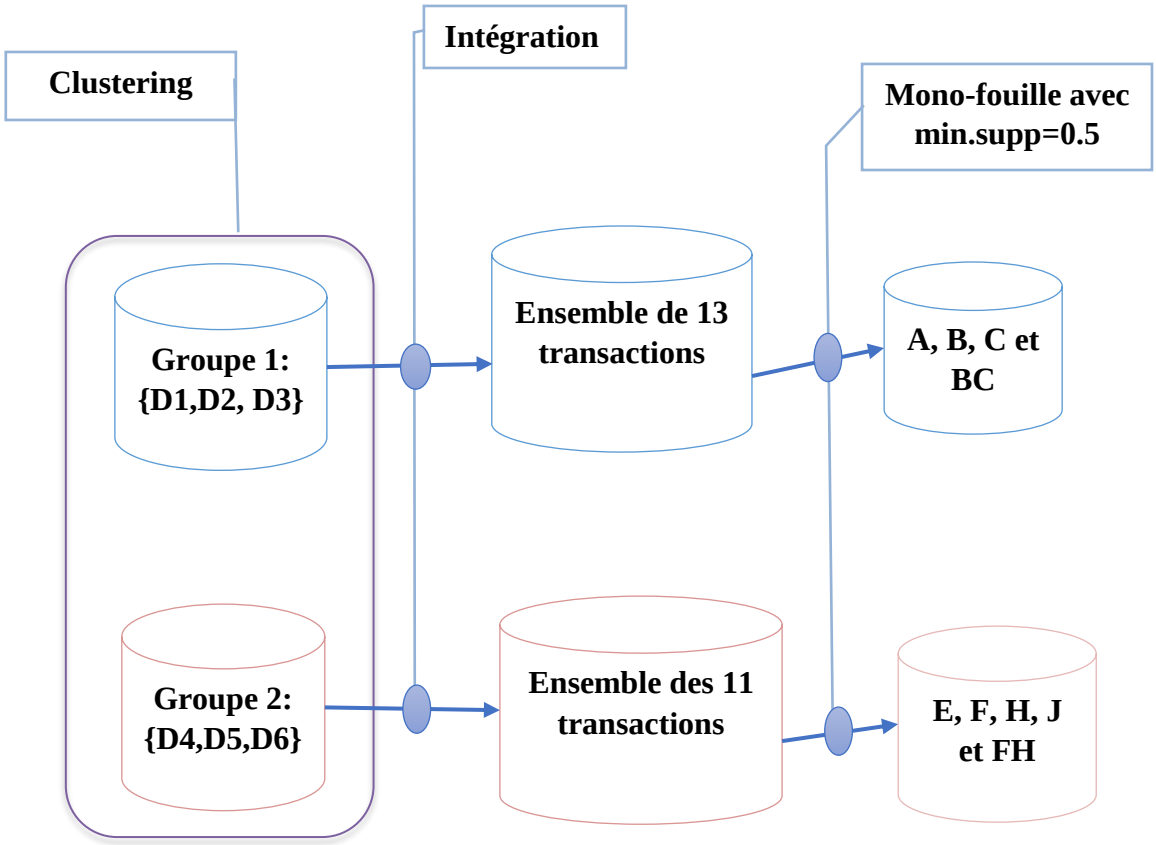



**FIGURE 2.5 -** APPLICATION DE LA FOUILLE MONOBASE DE DONNEES SUR CHAQUE CLASSE DE BASE DE DONNEES PERTINENTES

### 3.2.1 Méthode proposée :

La méthode de classification proposée se base sur :

- La mesure de la pertinence des bases de données indépendamment de toute application et
- La recherche et l'évaluation de la meilleure classification minimisant une fonction objective.

Pour décrire leur approche, les auteurs [Wu et al. 2005] se sont basés sur les définitions suivantes :

**a) Définition de la mesure de similarité :**

Les auteurs [Wu et al. 2005] ont proposé d'utiliser les items de données comme vecteur d'attributs pour décrire une base de données transactionnelle. Donc, si deux bases de données partagent un nombre important d'items en communs, alors ces deux bases sont similaires ou pertinentes.

La sélection d'attributs dans une base de données n'est pas une tâche facile. Puisque les bases de données du monde réel sont souvent très volumineuses, il est coûteux de chercher tous les items dans toutes les bases. Dans ce cas, les techniques d'échantillonnage [Toivonen et al. 1996] peuvent être utilisées sur chaque base de données volumineuse**.**

Soient *D1,D2,..Dm*, les *m* bases de données des *m* succursales d'une entreprise multi-branches. Items(*Di*) est l'ensemble des items dans *Di* (*i=1,2,…,m*). La similarité $sim_1$ entre les deux bases de données *Di* et *Dj* est définie comme suit :

$$sim1(Di, Dj) = \frac{|Items(Di) \cap Items(Dj)|}{|Items(Di) \cup Items(Dj)|}$$

Ou " ∩ " dénote l'ensemble d'intersection, " ∪ " dénote l'ensemble de l'union et "|Items(Di) ∩ Items(Dj)|" est le nombre d'éléments dans l'ensemble Items(Di) ∩ Items(Dj).

Deux bases de données avec un grand ensemble d'intersection possèdent un grand degré de similarité. Dans le cas contraire, elles sont considérées plutôt dissimilaires. Une autre mesure $sim_2$ a été définie et qui prend en compte les items issus des règles d'associations extraites à partir de chaque base.

**b) Définition de la mesure *Goodness* d'une classification :**

Afin d'évaluer une classification des multi-bases de données, une mesure *Goodness* a été définie pour mesurer la qualité de la classification générée.

Soit $class(D,sim_1,\alpha)=\{class^{\alpha}_1, class^{\alpha}_2,\ldots, class^{\alpha}_n\}$ une classification des *m* bases de données sous le seuil de similarité $\alpha \in [0,1]$.

La qualité d'une classification *class* est définie comme suit :

$$Goodness(class, \alpha) = \sum_{i=1}^{n} Value(class_i^{\alpha})$$

Où

$$Value(class_k^{\alpha}) = \sum_{Di,Dj \in class^{\alpha}}^{i \neq j} (1 - sim1(Di, Dj))$$

La valeur (1- $sim_1$(*Di*,*Dj*)) est la distance entre deux bases de données *Di* et *Dj* via la mesure $sim_1$.

Si la distance est faible alors cela correspond à un grand degré de similarité, sinon une grande distance signifie que les deux bases de données sont plutôt dissimilaires.

*Value*(*class*$^{\alpha}_{k}$) représente la somme des distances de chaque paire de bases de données dans *class*$^{\alpha}_{k}$ via la mesure $sim_1$.

On dit que *class*(*D*,$sim_1$,$\alpha$) est *la meilleure classification* obtenue des *m* bases de données si *distance*$^{f}_{Goodness}$ (*class,* $\alpha$) retourne la valeur minimale sous le seuil de similarité α tel que :

$$distance^{f}_{Goodness}(class, \alpha) = |Goodness(class, \alpha) - |class||$$

Où |*class*| est le nombre de clusters dans la classification générée.

### 3.2.2 Complexité de l'algorithme [Wu et al. 2005] :

Pour chercher la meilleure classification des *m* multiples bases de données, les auteurs ont proposé une approche à deux étapes. La première étape consiste à générer une classification pour un seuil $\alpha$ donné. La seconde étape consiste à évaluer la meilleure classification via la mesure *Goodness*. Ces deux étapes s'exécutent l'une après l'autre en faisant décrémenter le seuil de similarité $\alpha$ d'un pas $\lambda$, $\alpha = \alpha - \lambda$. L'algorithme de génération et recherche de la meilleure classification *BestClassification* est illustré dans la figure 2.6.

La complexité temporelle de l'algorithme est $O(hm^4)$, où *m* est le nombre de bases de données et $h = Int(1/\lambda)$ est une fonction de la division entière de 1 sur le pas $\lambda$

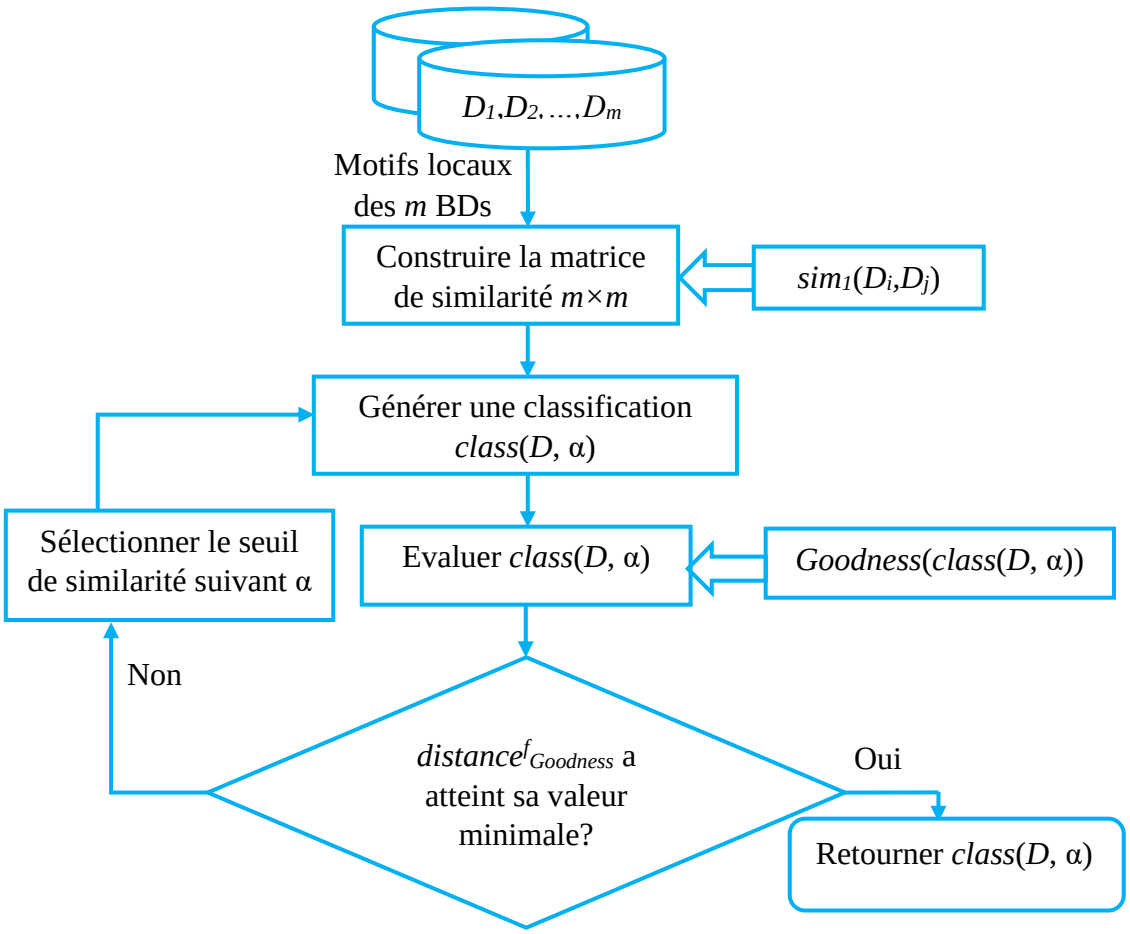



**FIGURE 2.6 -** ALGORITHME BESTCLASSIFICATION DE RECHERCHE DE LA MEILLEURE CLASSIFICATION DES M BASES DE DONNEES [WU ET AL. 2005].

### 3.3 Algorithme de classification de bases de données *amélioré* pour la fouille multi-bases de données [Li et al. 2009]

Les auteurs [Li et al. 2009] ont présenté une approche de classification des multi-bases de données qui améliore la complexité temporelle de l'algorithme *BestClassification* proposé par [Wu et al. 2005]**.** Pour cela, [Li et al. 2009] se sont basés sur les observations suivantes :

L'algorithme *BestClassification* [Wu et al. 2005] utilise un *pas* de recherche $\lambda$ pour trouver la meilleure classification ayant une valeur minimale *Distance*($\alpha$) (métrique d'évaluation de la meilleure classification proposée par [Wu et al. 2005]**)** . Si $\lambda$ est très petit, alors la meilleure classification peut être obtenue après un temps d'exécution plus long, car le seuil de similarité $\alpha$ avance avec des petits *pas*. Si $\lambda$ est très grand, *BestClassification* peut rater la meilleure classification en sautant la valeur minimale de la fonction *Distance*($\alpha$).

En analysant ces limites, l'algorithme *BestClassification* a été ajusté et amélioré par [Li et al. 2009] afin d'optimiser sa complexité temporelle et éviter de manquer la meilleure classification des *m* multi-bases de données.

### 3.3.1 Méthode proposée :

La méthode utilisée est toujours basée sur les mêmes objectifs cités par [Wu et al. 2005]**,** à savoir :

1. Mesurer la pertinence des bases de données indépendamment de toute application.
2. Chercher et évaluer la meilleure classification en sélectionnant efficacement les seuils de similarité.

La même mesure de similarité $sim_1$ et les concepts $Value(class^{\alpha})$, $Goodness(\alpha)$ *et* $Distance(\alpha)$ cités dans [Wu et al. 2005] ont été réutilisés afin de vérifier la logique des résultats de classification obtenus. Afin de ne pas manquer la meilleure classification en choisissant un pas non approprié, les auteurs [Li et al. 2009] ont utilisé comme seuil de similarité $\alpha$, l'ensemble des valeurs de similarités distinctes entre les bases. Ces valeurs distinctes issues de la table de similarité sont d'abord triées dans un ordre croissant puis utilisées comme seuil de similarité.

### 3.3.2 Complexité de l'algorithme [Li et al. 2009] :

Comme dans l'approche précédente [Wu et al. 2005], les auteurs ont proposé un algorithme à deux étapes. La première étape consiste à générer une classification pour un seuil de similarité $\alpha$ choisi dans l'ordre croissant des valeurs de similarités distinctes entre les bases. La seconde étape consiste à rechercher la meilleure classification minimisant une certaine mesure *goodness.*

Soit *h* le nombre des valeurs de similarités distinctes entre les *m* bases de données, alors la complexité temporelle de *BestCompleteClass* [Li et al. 2009] est $O(hm^3)$ ce qui est beaucoup mieux comparé à *BestClassification* [Wu et al. 2005]**.**

## 3.4 Clustering des bases de données basé sur les motifs locaux [Adhikari et al. 2010a]

Les auteurs [Adhikari et al. 2010a] ont amélioré le processus de clustering des multi-bases de données en se basant sur les stratégies suivantes :

- Réduction du temps d'exécution de l'algorithme de clustering et
- Utilisation d'une mesure de similarité basée sur les motifs locaux.

### 3.4.1 Méthode proposée :

Soit *n* le nombre de bases de données et soit FIS($D_i$, $\alpha$) l'ensemble des itemsets fréquents correspondant à une base de données $D_i$ sous le support minimum $\alpha$.

Le clustering des *n* bases de données suit les étapes suivantes :

1. Trouver FIS($D_i$, $\alpha$), pour i=1,2,…,n
2. Déterminer la similarité entre chaque paire de base de données en utilisant la mesure proposée $simi_2$
3. Calculer la mesure de qualité *goodness* pour les classifications non triviales trouvées et
4. Retourner la classification pour laquelle la valeur *goodness* est maximale.

Pour exécuter les étapes précédentes, les auteurs de cette approche se sont basés sur les métriques suivantes.

#### a) Définition de la mesure de similarité :

La mesure de similarité $sim_2$ entre deux bases de données Di et Dj est définie comme suit :

$$sim_2(D_i, D_j, \alpha) = \frac{\sum_{X \in (FIS(D_i,\alpha) \cap FIS(D_j,\alpha))} min\{ supp(X,D_i), supp(X,D_j) \}}{\sum_{X \in (FIS(D_i,\alpha) \cup FIS(D_j,\alpha))} max\{ supp(X,D_i), supp(X,D_j) \}}$$

Tel que $\alpha$ représente le support minimum de fouille des itemsets fréquents dans chaque base, FIS($D_i$, $\alpha$) représente l'ensemble des itemsets fréquents extraits à partir de $D_i$ et supp(X,$D_i$) représente le support local de l'itemset X dans $D_i$.

Pour le numérateur, on prend la somme des supports minimaux des itemsets fréquents communs aux deux bases de données Di et Dj. Pour le dénominateur, on prend la somme des supports maximaux de tous les itemsets fréquents des deux bases de données Di et Dj.

**b) Définition du critère d'évaluation d'une classification :**

La meilleure classification est basée sur le principe de maximisation de la similarité intra-classe et la maximisation de la distance inter-classe. La similarité intra-classe intra-sim d'une classification π à un niveau de similarité δ est définie comme suit :

$$intra\_sim(\pi_2^{\delta}) = \sum_{C \in \pi_2^{\delta}} \sum_{Di,Dj \in C,\, i<j} sim2(Di, Dj, \alpha)$$

La distance inter-classe inter-sim d'une classification π à niveau de similarité δ est définie comme suit :

$$inter\_dist(\pi_2^{\delta}) = \sum_{C_p,C_q \in \pi_2^{\delta};\, p<q} \sum_{Di \in C_p; Dj \in C_q,\, i<j} (1 - sim2(Di, Dj, \alpha))$$

La meilleure classification de l'ensemble des classifications générées est sélectionnée en se basant sur la valeur de la mesure *goodness* :

$$goodness(\pi_2^{\delta}) = intra\_sim(\pi_2^{\delta}) + inter\_dist(\pi_2^{\delta}) - |\pi_2^{\delta}|$$

Tel que $|\pi_2^{\delta}|$ est le nombre de tous les clusters dans la classification π.

Plus la valeur *goodness* est grande, meilleure est la classification trouvée.

### 3.4.2 Complexité de l'algorithme [Adhikari et al. 2010a] :

Après avoir calculé et trié toutes les valeurs de similarité non-nulles entre les bases de données dans l'ordre décroissant, l'algorithme va générer une classification en démarrant avec la valeur de similarité maximale. Chaque classification est évaluée ainsi en utilisant la mesure *goodness*. Après la génération de toutes les *m* classifications possibles, la classification obtenant une valeur *goodness* maximale est choisie comme la meilleure classification des *n* multiples bases de données.

La complexité temporelle de l'algorithme *BestDatabasePartition* proposé par [Adhikari et al. 2010a] est $O(m \times n^2)$ où $1 \leq m \leq (n^2-n)/2$, ce qui est nettement mieux que ses prédécesseurs [Li et al. 2009] et [Wu et al. 2005].

## 3.5 Clustering complet pour la fouille multi-bases de données [Liu et al. 2013]

Dans cette approche, les auteurs se sont basés sur une mesure de similarité basée sur les itemsets fréquents comme dans [Adhikari et al. 2010a]. De plus, une nouvelle mesure d'évaluation de la qualité de la classification a été conçue. Un nouvel algorithme a été proposé aussi pour la génération d'une classification.

### 3.5.1 Méthode proposée :

La méthode proposée permet de générer toutes les classes possibles par niveau. Soit *n* bases de données à classer. Initialement au niveau 1, chaque classe contient une seule base de données. En se basant sur les classes du niveau 1, les classes du niveau 2 sont créées par la fusion de la i-ème classe contenant Di avec la j-ième classe contenant Dj si et seulement si la similarité entre Di et Dj est supérieure à δ, où δ est un seuil de similarité préfini. Le processus continue ainsi jusqu'à ce qu'il n'y ait plus de classes à générer.

**a) Définition de la mesure de similarité :**

La mesure de similarité utilisée est $sim_2$ définie dans [Adhikari et al. 2010a]**.**

**b) Définition du critère d'évaluation d'une classification :**

La meilleure classification est basée sur le principe de maximisation de la similarité intra-classe et la distance inter-classe.

**c) Définition de la similarité intra-classe *intra-sim* :**

La similarité intra-classe d'une classification $C^*$ est définie par la somme moyenne des similarités intra-classe comme suit :

$$intra\_sim(C^*) = \frac{1}{n}\sum_{t=1}^{n} sim(C_t)\ , for\ |C^*| = n,$$

$$\text{Où,} \qquad sim(C_t) = \begin{cases} 1, |C_t| = 1, \\ \dfrac{\sum_{Di,Dj \in C_t} sim(Di, Dj, \alpha)}{C_k^2}, |C_t| = k > 1, \end{cases}$$

Cette mesure exprime le degré de cohésion des bases au sein d'une même classe. Plus sa valeur est grande, plus la classification est meilleure.

**d) Définition de la distance inter-classe *inter-dist* :**

La distance inter-classe *inter-dist* d'une classification $C^*$ est définie par la somme moyenne des distances inter-classe comme suit :

$$inter\_dist(C^*) = \begin{cases} 0, |C^*| = 1, \\ \dfrac{\sum_{Cp,Cq \in C^*} \dfrac{\sum_{Di \in Cp, Dj \in Cq; i<j} \left(1 - sim(Di, Dj, \alpha)\right)}{p * q}}{C_n^2} \\ |C^*| = n > 1, |Cp| = p, |Cq| = q \end{cases},$$

Cette mesure exprime le degré de corrélation entre les classes d'une même classification. Plus sa valeur est grande, plus les classes dont disjointes et éloignées et donc plus la classification est optimale.

**e) Définition de la mesure de qualité d'une classification *goodness* :**

$$goodness(C^*) = \frac{intra\_sim(C^*) + inter\_dist(C^*)}{n}$$

Cette mesure se base sur les deux mesures précédentes et le nombre de clusters dans la classification. La classification $C^*$ est dite optimale si elle possède une plus grande valeur *goodness*.

### 3.5.2 Complexité de l'algorithme [Liu et al. 2013] :

L'algorithme proposé génère toutes les classes possibles par niveau comme illustré dans la figure 2.7. Il est simple à implémenter mais le clustering réalisé n'est pas efficace, en particulier quand le nombre de bases de données devient grand.

La complexité temporelle de l'algorithme est exponentielle, de l'ordre $O(t \times 2^n \times n)$, tel que $n$ est le nombre de bases de données et $t$ est le nombre de valeurs de similarités distinctes entre les $n$ bases de données.

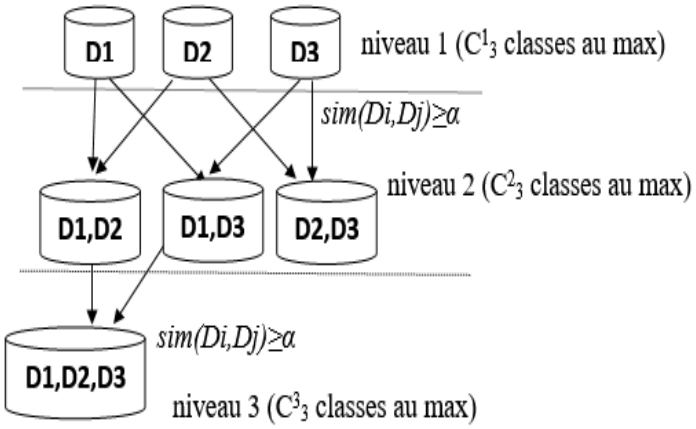



**FIGURE 2.7 -** APPROCHE DE GENERATION DES CLUSTERS PAR NIVEAU

## 3.6 La classification multi-relationnelle [Yin and Han 2005]

Le but de la classification multi-relationnelle est de construire un modèle précis permettant de prédire les étiquettes de classe des nouveaux exemples de la relation cible étant donné des exemples pré-classés. Pour apprendre les descriptions des classes, ces approches essaient d'utiliser le minimum de connexion inter-relationnelles pour une prédiction optimale des étiquètes de classes des tuples cibles.

La classification multi-relationnelle ne peut être utilisée pour le clustering des multi-bases de données car dans notre thèse il s'agit d'une classification non supervisée où des clusters de bases de données sont découverts à partir de l'analyse des similarités entre les bases de données.

## 4 Synthèse et étude critique

Dans cette section nous présentons une synthèse et une étude critique des approches et algorithmes de classification étudiés.

### a) Algorithme [Liu et al. 2001] [Liu et al. 1998] :

Les auteurs [Liu et al. 2001] [Liu et al. 1998] ont proposé une approche pour identifier les bases de données les plus pertinentes à une requête utilisateur *Q*. Pour cela une mesure de pertinence *RF* a été proposée ayant comme objectif de mesurer la déviation entre la requête utilisateur *Q* et les attributs d'une table relationnelle. Un algorithme de sélection des bases de données pertinentes a été conçu avec une complexité temporelle de l'ordre $O(m \times N \times M)$, où *N* est le nombre d'enregistrements dans la base, *M* est le nombre d'attributs et *m* est le nombre des bases de données à analyser.

Bien que la méthode proposée fournit de bons résultats en sélectionnant les bases les plus pertinentes à une requête donnée, la recherche exhaustive de cette méthode dans toute la base est non faisable, en particulier lorsque la base de données est très volumineuse. Effectuer une recherche aléatoire telle que celle proposée par [Motwani and Raghavan 2010] peut être une voix possible à explorer.

### b) Algorithme [Wu et al. 2005] :

La méthode proposée par [Liu et al. 2001] [Liu et al. 1998] représente un moyen efficace pour éviter la jointure forcée des bases de données non pertinentes avec celles qui sont pertinentes à la requête utilisateur. En outre, elle permet de réduire l'espace de recherche et améliorer la qualité des connaissances découvertes. Cependant, cette méthode est dépendante de l'application de la fouille de données et donc ne peut être utilisée pour la fouille multi-bases de données sans spécifier une requête de l'utilisateur.

Cela a motivé [Wu et al. 2005] à concevoir une stratégie de classification des bases de données *indépendante* de toute application dans un scenario de fouille multi-bases de données. Ce type de classification consiste à utiliser une mesure de similarité afin de regrouper les bases de données les plus similaires au sein d'une même classe. Pour cela une mesure de similarité basée sur les items partagés entre les bases a été définie. En faisant varier un seuil de similarité, différentes classifications peuvent être formées.

La complexité temporelle de l'algorithme de classification proposé par [Wu et al. 2005] est de l'ordre $O(h \times m^4)$, où $m$ est le nombre de bases de données et $h$ est le nombre de classifications générées avant d'obtenir la meilleure classification.

La mesure de similarité définie par [Wu et al. 2005] est basée sur les items dans les bases. Cela peut être utile pour estimer la similarité entre deux bases de données volumineuses. Cependant, cette mesure reste estimative et inadéquate car dans la réalité, deux bases de données sont similaires si elles ont plusieurs transactions en commun et deux transactions sont similaires si elles ont plusieurs items partagés. Donc, une mesure de similarité efficace peut être conçue en se basant sur les itemsets fréquents et leurs supports dans les bases de données et non pas sur les items. Une telle mesure peut être utile pour savoir si des clients ont le même comportement d'achat dans les différentes succursales.

Bien que la classification des multi-bases de données proposée par [Wu et al. 2005] est satisfaisante, le choix arbitraire du seuil de similarité et la complexité temporelle de l'algorithme pose toujours un problème lorsque le nombre de bases de données devient grand. De plus, l'algorithme proposé échoue dans certains cas à trouver la meilleure classification en choisissant un pas de similarité non approprié et dans certains cas il boucle indéfiniment lors de la recherche d'une classification.

**c) Algorithme [Li et al. 2009] :**

Les travaux ci-dessus ont motivé les auteurs [Li et al. 2009] à ajuster et modifier l'algorithme proposé par [Wu et al. 2005] afin d'optimiser sa complexité temporelle et obtenir une meilleure classification des $m$ bases de données. Ainsi la même mesure de similarité et les concepts $Value(class^{\alpha})$, $Goodness(\alpha)$ *et* $Distance(\alpha)$ définis par [Wu et al. 2005] ont été réutilisés. Afin de ne pas rater la meilleure classification en choisissant un *pas* non approprié, les auteurs ont utilisé comme seuil de similarité, l'ensemble des valeurs de similarités distinctes entre les bases de données. Ces valeurs distinctes issues de la table de similarité sont d'abord triées dans un ordre croissant puis utilisées comme seuil de similarité.

L'algorithme de classification proposé a été optimisé et sa complexité temporelle est de l'ordre $O(h \times m^3)$, tel que $m$ est le nombre de bases de données et $h$ est le nombre de classifications générées avant d'obtenir la meilleure classification.

Bien que l'approche proposée est efficace et trouve toujours la meilleure classification, la mesure de similarité utilisée reste non appropriée puisque elle se base uniquement sur les

items communs entre les bases. En outre, la complexité temporelle de l'algorithme peut être optimisée encore plus.

**d) Algorithme [Adhikari et al. 2010a] :**

Afin d'améliorer la complexité temporelle et la mesure de similarité des algorithmes proposés par [Li et al. 2009] [Wu et al. 2005]**,** les auteurs [Adhikari et al. 2010a] ont proposé un nouvel algorithme de clustering des multi-bases de données basé sur l'optimisation du temps d'exécution de l'algorithme de clustering, l'utilisation d'une mesure de similarité plus appropriée et un stockage efficace des itemsets fréquents en mémoire centrale. En effet, [Adhikari et al. 2010a] ont amélioré la mesure de similarité entre les bases en faisant intervenir les itemsets fréquents ainsi que leurs supports. La complexité temporelle de l'algorithme proposé par [Adhikari et al. 2010a] est de l'ordre $O(m \times n^2)$, tel que $n$ est le nombre de bases de données et $m$ est le nombre de similarités distinctes entre les multi-bases de données.

Afin d'augmenter la précision de la mesure de similarité, [Adhikari et al. 2010a] ont proposé une approche de codage efficace pour la représentation des itemsets fréquents en mémoire centrale. Cela va permettre de réduire la valeur du support minimum afin de stocker plus d'itemsets fréquents en mémoire centrale. Ainsi, le processus de clustering va devenir plus précis.

Bien que l'approche proposée est efficace et permet d'avoir des classifications correctes et optimales, l'algorithme proposé peut devenir lent lorsque le nombre de bases de données augmente. En outre, la mesure de similarité proposée ne prend pas en charge l'apport des itemsets non fréquents qui peuvent grandement améliorer la synthèse des itemsets globaux.

**e) Algorithme [Liu et al. 2013]:**

Cet algorithme génère toutes les classes possibles des bases de données par niveau. Donc, au début il génère toutes les classes de taille 1 puis toutes les classes de taille 2 et ainsi de suite jusqu'à générer une classe de toutes les bases de données. En tout, $2^n$ classes sont générées.

Cet algorithme est simple à implémenter. Cependant, sa limite réside dans sa complexité exponentielle qui peut rendre le code très lent lorsque le nombre de bases de données devient grand. De plus, la mesure de similarité utilisée pend en considération uniquement les items dans les bases, ce qui n'est pas précis pour calculer la similarité entre les bases de données.

### 4.1 Tableau comparatif

Le tableau 2.1 suivant résume les limites et les points forts des algorithmes de clustering étudiés.

| Algorithme | Complexité temporelle | Avantages | Limites |
|---|---|---|---|
| **[Liu et al. 2001]** | $O(m \times N \times M)$, où<br><br>$N$ **:** nombre max d'enregistrements dans les $m$ bases de données.<br><br>$M$ **:** nombre max d'attributs dans les $m$ bases de données.<br><br>$m$ **:** nombre de bases de données relationnelles. | • Evite la jointure forcée des bases de données non pertinentes avec celles pertinentes.<br><br>• Réduit l'espace de recherche à explorer.<br><br>• Améliore la découverte des motifs pertinents. | • Dépendant de la requête de l'utilisateur.<br><br>• Ne peut être utilisé pour le clustering des multi-bases de données. |
| **[Wu et al. 2005]** | $O(h \times m^4)$, où<br><br>$h$**:** nombre max de classifications générées.<br><br>$m$ **:** nombre de bases de données transactionnelles. | • Indépendant de la requête de l'utilisateur.<br><br>• Permet la découverte et l'analyse des clusters de bases de données. | • Mesure de similarité basée sur les items (qualité des motifs synthétisés réduite).<br><br>• Echoue à trouver la meilleure classification lorsque un *pas* de recherche non adéquat est assigné.<br><br>• Complexité temporelle élevée :<br><br>$O(h \times m^4)=O(m^6)$ au pire des cas |

| | | | |
|---|---|---|---|
| **[Li et al. 2009]** | $O(h \times m^3)$, où<br><br>***h*:** nombre max de classifications générées.<br><br>***m* :** nombre de bases de données transactionnelles. | • Choix approprié du *pas* de recherche de la meilleure de classification.<br><br>• Réduction de la complexité temporelle. | • Mesure de similarité basée sur les items (qualité des motifs synthétisés réduite).<br><br>• Complexité temporelle reste à améliorer : $O(h \times m^3)=O(m^5)$ au pire des cas |
| **[Adhikari et al. 2010a]** | $O(m \times n^2)$),où<br><br>***m*:** nombre max de classifications générées.<br><br>***n*:** nombre de bases de données transactionnelles. | • Choix approprié du *pas* de recherche de la meilleure de classification.<br><br>• Mesure de similarité basée sur les supports des itemsets fréquents.<br><br>• Réduction de la complexité temporelle. | • Mesure de similarité ignorant les itemsets non fréquents.<br><br>• Complexité temporelle reste à améliorer :<br><br>$O(m \times n^2)=O(n^4)$ au pire des cas |
| **[Liu et al. 2013]** | $O(t \times 2^n \times n)$, où<br><br>***t*:** nombre de valeurs de similarités distinctes entre les *n* bases de données.<br><br>***n*:** nombre de bases de données transactionnelles. | • Simple à implémenter. | • Mesure de similarité basée sur les items (qualité des motifs synthétisés réduite).<br><br>• Complexité temporelle exponentielle :<br><br>$O(t \times 2^n \times n)= O(n^3 \times 2^n)$, au pire des cas. |
| **[Miloudi et al. 2018]**<br><br>**[Miloudi et al. 2016a]** | $O(n^2)$, où<br><br>***n*:** nombre de bases de données transactionnelles | • Réduction de la complexité temporelle.<br>• Amélioration de mesure de similarité. | • Basé sur la mesure ***goodness* [Adhikari et al. 2010a]** qui est non-convexe et donc difficile à optimiser. |

**Table 2.1 –** Tableau de synthèse des algorithmes de clustering des multi-bases de données.

## 5 Conclusion

Dans ce chapitre nous avons étudié les approches de classification des multi-bases de données existantes tout en mettant l'accent sur les algorithmes utilisés, leurs points forts et leurs limites. Une synthèse et une étude critique ont été réalisées en se basant sur une comparaison effectuée entre les différentes approches.

Afin d'améliorer la performance des algorithmes de clustering des multi-bases de données, nous nous sommes basés sur les deux points suivants :

1. La réduction de la complexité temporelle de ces algorithmes et
2. l'amélioration de la mesure de similarité entre les bases de données.

Pour réduire la complexité temporelle des algorithmes de clustering proposés par [Wu et al. 2005], [Li et al. 2009] et [Adhikari et al. 2010a], nous allons proposer un algorithme [Miloudi et al. 2018] basé sur la découverte des sous-graphes complets de bases de données dans un graphe de similarité. L'utilisation d'une structure de données de type graphe permet de garder une trace des clusters de bases de données générés dans les classifications précédentes afin de les réutiliser pour générer de nouvelles classifications. L'analyse théorique, les exemples et les expérimentations présentées dans le chapitre 3 et 4 prouvent bien l'efficacité de l'algorithme proposé à trouver la meilleure classification des multi-sources de données en un temps plus court comparé aux algorithmes [Wu et al. 2005], [Li et al. 2009] et [Adhikari et al. 2010a].

La mesure de similarité proposée par [Adhikari et al. 2010a] se base uniquement sur les supports des itemsets fréquents partagés entre les bases de données. Donc, plus il y a un grand nombre d'itemsets fréquents en commun et ayant des valeurs de support proches, plus les bases de données sont considérées comme étant similaires. Cependant, cette mesure de similarité *ignore* les itemsets locaux non fréquents qui peuvent éventuellement exister dans les bases de données avec des supports plus au moins proches du seuil du support minimum. En effet, lors de la synthèse des itemsets globaux via les modèles définis par [Adhikari et al. 2010b] [Adhikari and Rao 2008] [Khiat 2015] [Ramkumar and Srinivasan 2010] [Ramkumar and Srinivasan 2008] [Wu and Zhang 2003] [Zhang et al. 2009] [Zhang and Zaki 2006], les itemsets non fréquents peuvent grandement contribuer dans l'estimation des

supports exacts des itemsets globaux. Par conséquent, nous proposons dans [Miloudi et al. 2016a] une mesure de similarité qui utilise le modèle de synthèse défini par [Ramkumar and Srinivasan 2008] et le facteur de correction *h* [Srinivasan and Ramkumar 2009] afin d'inclure la contribution des itemsets non fréquents dans le processus de synthèse des itemsets globaux. Des exemples expérimentaux sont présentés dans le chapitre 3 afin de montrer l'efficacité de la mesure de similarité proposée.

Dans le chapitre suivant, nous présentons notre contribution pour améliorer la performance des algorithmes de clustering des multi-sources de données.

# Partie II – Contribution

# Chapitre 3 -

# Amélioration de la performance du clustering des multi-bases de données

# 1 Introduction

Afin de trouver la meilleure classification d'un ensemble de *n* bases de données, quelques algorithmes ont été proposés dans la littérature [Adhikari et al. 2010a] [Li et al. 2009] [Liu et al. 2013] [Wu et al. 2005]. Le plus optimal d'entre eux en matière de précision et temps d'exécution est celui proposé par [Adhikari et al. 2010a]. Cet algorithme permet de trouver la meilleure classification d'un ensemble de *n* bases de données en un temps estimé à $O(m \times n^2)$, tel que $m$ $(1 \leq m \leq (n^2-n)/2)$ est le nombre total de classifications candidates générées.

Lorsque $m$ est égale à $(n^2-n)/2$ et $n$ est très grand, le temps d'exécution nécessaire pour la classification des multi-bases de données va largement augmenter. Ainsi, il devient important de proposer un algorithme de clustering optimal qui réduit la complexité temporelle du processus de classification existant. Dans ce chapitre, nous décrivons l'approche et les algorithmes proposés afin d'arriver à notre but.

# 2 Limites des travaux existants

A partir de l'étude conduite dans le chapitre 2, les algorithmes existants [Adhikari et al. 2010a] [Li et al. 2009] [Liu et al. 2013] [Wu et al. 2005] produisent de 1 à $(n^2-n)/2$ classifications hiérarchiques candidates qui vérifient la propriété 1. La figure 3.1 illustre l'imbrication automatique des clusters découverts à un niveau de similarité $\delta_i$ dans les clusters découverts à niveau de similarité suivant $\delta_{i+1}$.

**Propriété 1:**

Soit $class(D, \delta_i)$ et $class(D, \delta_{i+1})$ deux classifications candidates de l'ensemble $D=\{D_1,D_2,\ldots,D_n\}$ générées à deux niveaux de similarité consécutifs $\delta_i$ et $\delta_{i+1}$ respectivement. Alors on peut constater que pour chaque cluster $g_x$ dans $class(D,sim, \delta_i)$ , il existe un autre cluster $g_y$ dans $class(D,sim, \delta_{i+1})$, tel que $g_x$ est inclus dans $g_y$ (e.i., $g_x \subset g_y$)

En dépit de la propriété 1, les algorithmes existants produisent chaque classification indépendamment, sans tenir compte de la réutilisation des clusters générés auparavant. En effet, ils génèrent chaque classification en commençant de l'état initial où chaque base de données forme un cluster à un seul élément appelé singleton $\{D_1\},\{D_2\},\ldots,\{D_n\}$.

Par conséquent, les algorithmes existants sont coûteux en matière temps d'exécution et cumulent un travail additionnel inutile en générant à nouveau les clusters qui ont été déjà générés dans les classifications précédentes.

Afin de résoudre ce problème, nous proposons dans ce chapitre une approche de clustering qui représente le problème de classification des multi-bases de données $D=\{D_1,D_2,\ldots,D_n\}$ comme un problème d'identification des sous-graph complets dans un graphe de similarité pondéré non orienté $G=(D,E,W)$. L'intuition d'utiliser une telle structure de données est argumentée dans les sections suivantes.

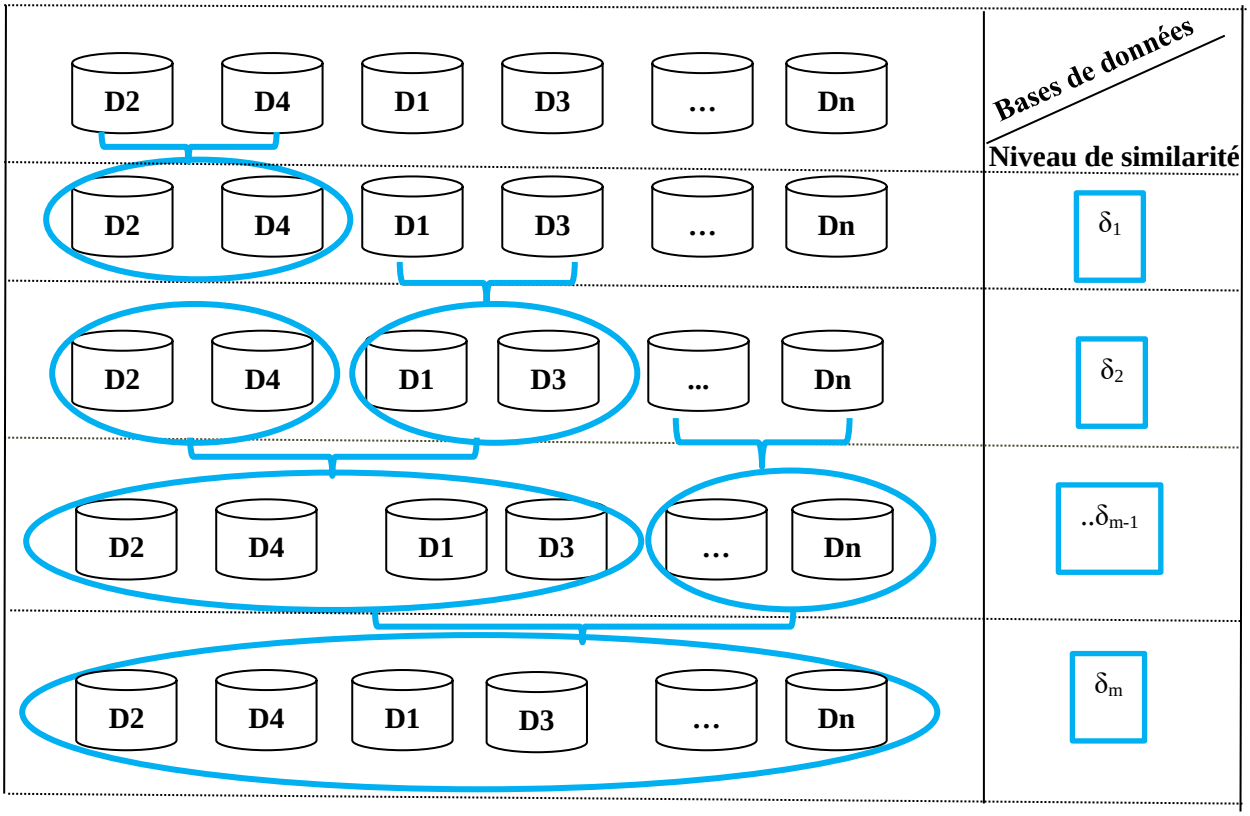



**FIGURE 3.1 -** LA PROPRIETE HIERARCHIQUE DES CLASSIFICATIONS GENEREES AUX DIFFERENTS NIVEAUX DE SIMILARITE.

## 3 Approche et Algorithme proposés

Dans cette section, nous décrivons l'approche proposée pour la classification des multi-bases de données. Nous supposons que chaque base de données a été déjà explorée pour la découverte des itemsets fréquents en utilisant l'un des algorithmes proposé par [Agrawal et al. 1994], [Agrawal and Shafer 1996], [Han et al. 2000], [Park et al. 1995].

- Soit $D=\{D_1,D_2,\ldots,D_n\}$ l'ensemble des $n$ bases de données à classer. Alors le problème de génération d'une classification $class(D,sim,\delta)$ à un niveau de similarité $\delta$, peut être décrit comme un problème de détermination des composantes connexes d'un graphe pondéré non orienté $G=(D,E,W)$ où $D$ est considéré comme l'ensemble des nœuds du graphe $G$ et $E$ est l'ensemble des arêtes et $W$ est l'ensemble des poids $w_{ij}$ des arêtes.

- Le poids $w_{ij}$ de chaque arête $(D_i,D_j) \in E$ est la valeur de similarité entre les correspondantes bases de données, notée $sim(D_i,D_j)$. A un certain niveau de similarité $\delta$, il existe une arrête connectant deux nœuds $D_i$ et $D_j$ , si et seulement si $sim(D_i,D_j) \geq \delta$. Dans ce qui suit, les termes *bases de données* et *nœuds* sont interchangeables.

- Initialement, le graphe $G=(D,E,W)$ est vide avec $n$ nœuds déconnectés (singletons) ,i.e., $|D|=n$ et $E=\emptyset$. Les valeurs de similarités sont calculées entre chaque paires de nœuds $(D_i,D_j)$ en utilisant une mesure de similarité appropriée, pour $i<j=1$ to $n$. Puis des listes d'arêtes sont ajoutées progressivement à $E$ en commençant par les arêtes ayant le plus grand poids comme suit :

➢ Soit $\delta_1,\delta_2,\ldots,\delta_m$ les $m$ valeurs de similarité distinctes entre les $n$ bases de données triées dans l'ordre décroissant (e.i., $\delta_1>\delta_2>\ldots>\delta_m$ ) et soit $E_{\delta l}=\{(D_i, D_j) \in E,\ sim(D_i,D_j)= \delta_l, i,j=1 \text{ to } n, i\neq j\}$ la liste des arêtes ayant la même valeur de poids $\delta_l$ ($l=1$ to $m$). Pour chaque valeur de similarité distincte $\delta_l$, une classification candidate $class(D,sim,\delta_l)$ est générée en ajoutant la liste $E_{\delta l}$ à l'ensemble des arêtes $E$ tel que $E=E_{\delta 1} \cup E_{\delta 2} \cup \ldots \cup E_{\delta l-1}$ (avec $l$-1 classifications générées auparavant). Puis, il reste juste à identifier les composantes connexes du graphe $G$ afin de trouver les clusters de bases de données dans $class(D,sim,\delta_l)$.

➢ A un niveau de similarité $\delta_l$, si chaque composante connexe de $G$, notée, $g_k^{\delta_l}$, forme une clique (e.i., un sous-graphe où chaque paire de nœuds est connectée par une arête dans $E$), alors la correspondante classification $class(D,sim,\ \delta_l)=\{\ g_1^{\delta_l},\ g_2^{\delta_l},\ \ldots,\ g_k^{\delta_l}\ \}$ est appelée classification *complète* . Dans cette thèse, nous nous intéressons à trouver de telles classifications.

- Pour l'évaluation de la classification, nous avons utilisé une mesure, notée *goodness*, basée sur la similarité intra-cluster, la distance inter-cluster et le nombre de clusters générés. La classification complète ayant une valeur *goodness* maximale est sélectionnée comme la meilleure classification de l'ensemble des bases de données. En se basant sur la mesure de similarité *sim* proposée dans [Miloudi et al. 2016a] , notre approche de classification est illustrée dans la figure 3.2.

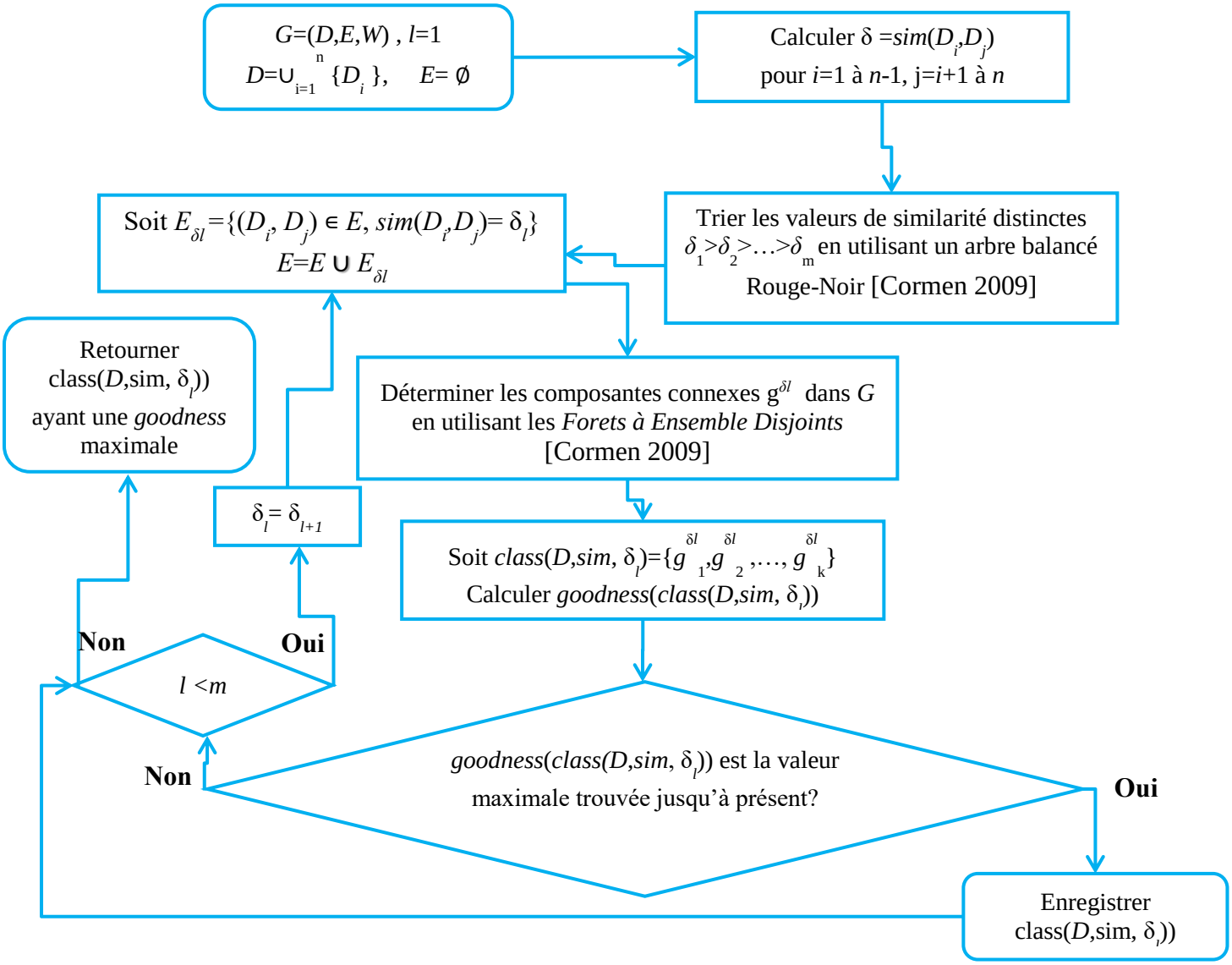



**FIGURE 3.2 -** APPROCHE PROPOSEE POUR LA CLASSIFICATION DES N MULTI-BASES DE DONNEES [MILOUDI ET AL. 2018].

Dans les sections suivantes nous présentons quelques définitions et exemples afin de bien comprendre l'approche proposée.

### 3.1 Concepts pertinents :

Inspirés par les travaux réalisés par [Adhikari et al. 2010a], [Ramkumar and Srinivasan 2008] et [Srinivasan and Ramkumar 2009] nous présentons les définitions suivantes.

**Définition 1 :**

Soit $D_i$ and $D_j$ deux bases de données transactionnelles et $\alpha$ la valeur du seuil du support minimum. Donc, la mesure de similarité entre deux bases de données $D_i$ et $D_j$ est définie comme suit :

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

Tel que,

$$\phi\{X, D_i \cup D_j\, , \alpha\} = \begin{cases} supp_s(X, D_i \cup D_j)\, , si\ supp_s(X, D_i \cup D_j) \geq \alpha \\ 0, sinon \end{cases}$$

Et,

$$supp_s(X, D_i \cup D_j) = \frac{supp(X,D_i)\times|D_i| + supp(X,D_j)\times|D_j|}{|D_i| + |D_j|}$$

Et,

$$supp(X, D_i) = \begin{cases} h \times \alpha\, , si\ supp(X, D_i) < \alpha \\ supp(X, D_i), sinon \end{cases}$$

$supp_s(X, D_i \cup D_j)$ représente le support synthétisé de $X$ dans $\{D_i \cup D_j\}$ tel que défini par [Ramkumar and Srinivasan 2008] et $supp(X,D_i)$ est le support local de $X$ dans $D_i$. En outre, nous notons par $|D_i|$ le nombre de transactions dans $D_i$ et $h \in [0,1]$ est le facteur de correction défini dans [Srinivasan and Ramkumar 2009].

La fonction $sim(D_i,D_j)$ calcule la similarité entre $D_i$ et $D_j$ en utilisant les supports synthétisés des itemsets globaux obtenus par la fouille de de l'union $D_i \cup D_j$. Plus de détails avec des exemples sont donnés dans la section 4 pour démontrer l'efficacité de cette mesure de similarité.

La mesure de similarité *sim* vérifie les égalités suivantes :

**Propriété 2** :

- $0 \leq sim(D_i,D_j, \alpha) \leq 1$
- $sim(D_i,D_j, \alpha) = sim(D_j,D_i, \alpha)$
- $sim(D_i,D_i, \alpha) = 1$ for $i,j=1...n$

**Définition 2 :**

La mesure distance *dist* entre deux bases de données $D_i$ et $D_j$ est définie comme suit :

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

La fonction distance *dist* vérifie les propriétés suivantes :

**Propriété 3** :

- $0 \leq dist(D_i,D_j, \alpha) \leq 1$
- $dist(D_i,D_j, \alpha) = dist(D_j,D_i, \alpha)$
- $dist(D_i,D_i, \alpha) = 1$ for $i,j=1...n$

**Définition 3 :**

Soit $class(D,sim,\delta)=\{g^{\delta}_1, g^{\delta}_2,...,g^{\delta}_k\}$ une classification candidate à $k$ clusters de l'ensemble de $D=\{D_1,D_1,...,D_n\}$ à un niveau de similarité $\delta$. La similarité intra-cluster de $class(D,sim,\delta)$ est définie comme suit :

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

**Définition 4 :**

La distance inter-cluster de $class(D,sim,\delta)$ à un niveau de similarité $\delta$ est définie comme suit :

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

La meilleure classification est sélectionnée en se basant sur la maximisation de la similarité intra-cluster et la distance inter-cluster.

**Définition 5 :**

La mesure de la qualité d'une classification *class*(*D,sim,δ*) est définie comme suit :

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

La classification qui obtient une valeur *goodness* maximale est sélectionnée comme la meilleure classification.

**Définition 6 :**

Soit $G=(D,E,W)$ le graphe représentant les clusters de bases de données alors $D=\{D_1, D_2,\ldots,D_n\}$ représente l'ensemble des nœuds dans le graphe $G$ et $E=\{( D_i, D_j), 1\leq i,j\leq n\}$ est l'ensemble des paires $(D_i, D_j)$ représentant l'ensemble des arêtes dans $G$.

**Définition 7 :**

A un niveau de similarité $\delta$, une classification *class*(*D,sim,δ*) est complète si et seulement si l'équation suivante est vérifiée :

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

Où $|g_i^\delta|$ représente le nombre de nœuds dans le sous-graphe $g_i^\delta$ et $|E|$ c'est le nombre d'arêtes dans le graphe $G$. Donc, si chaque cluster $g_i^\delta$ ($i$=1 to $k$) dans *class*(*D,sim,δ*) est une clique dans $G$ alors *class*(*D,sim,δ*) est une classification complète.

**Exemple :**

Soit $D=\{D_1, D_2,\ldots, D_6\}$ l'ensemble de base de données à classer. La table 3.1 représente la matrice des valeurs de similarité entre les six bases de données.

| ***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 |

**Table 3.1 –** La table de similarité des six bases de données.

- A partir de la table de similarité, il existe 5 valeurs de similarité distinctes. Par conséquent, il y a en tout 5 listes d'arêtes ayant le même poids. Les 5 valeurs de similarité distinctes sont listées dans l'ordre décroissant comme suit (0.79, 0.56, 0.50, 0.45, 0.40).

- Soit $G=(D,E,W)$ avec $|D|=6$ et $E=\emptyset$. Donc initialement on a une classification complète triviale $class(D)=\{\{D_1\},\{D_2\},\{D_3\},\{D_4\},\{D_5\},\{D_6\}\}$, avec $goodness(class(D))= 2.20$

- Pour chaque valeur de similarité $\delta$ sélectionnée dans l'ordre décroissant, une classification candidate $class(D,sim,\delta)$ est générée et testée si elle est complète comme suit :

  Pour $\delta=0.79$, on a, $E_{0.79}=\{(D_2, D_4)\}$, $E=E \cup E_{0.79}$ et donc
  $class(D,sim,0.79)=\{\{D_2,D_4\},\{D_1\},\{D_3\},\{D_5\},\{D_6\}\}$ avec $goodness(class(D,0.79))=3.78$

- $class(D,sim,0.79)$ est une classification complète car :

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

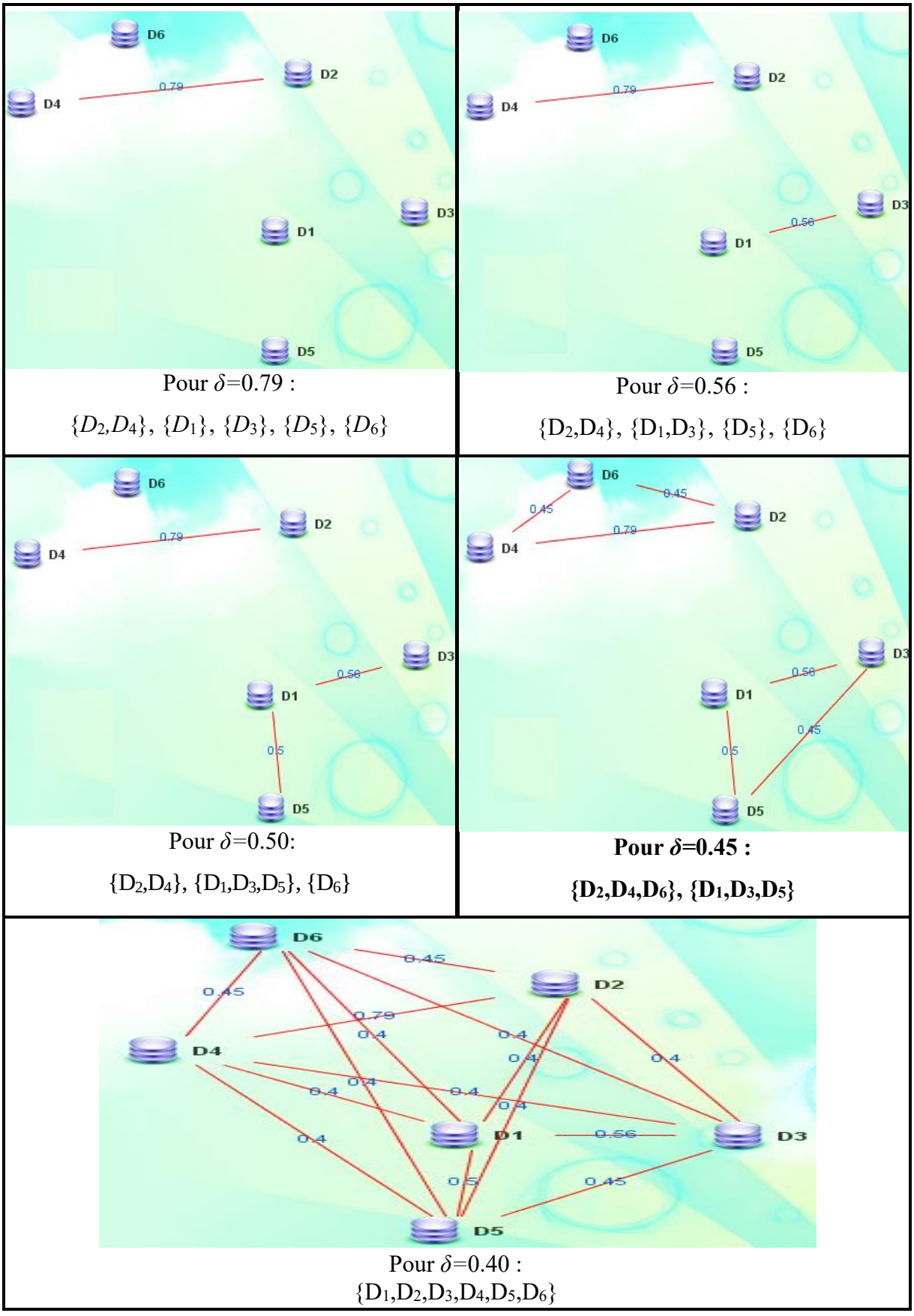
Les classifications restantes sont présentées dans la table 3.2 et la figure 3.3.

| **Niveau de similarité (δ)** | **Ensemble d'arêtes ($E_\delta$)** | **Classification candidate *class*(*D*,*sim*,δ)** | ***goodness*(*class*(*D*,δ))** |
|---|---|---|---|
| 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 |

**Table 3.2 –** Les classifications candidates générées à partir de la table de similarité **Table 3.1**

- Il existe une seule classification qui n'est pas complète au niveau de $\delta$=0.50 car $|E|$=3 et $\sum_{i=1}^{k} \frac{|g_i^{0.50}|^2 - |g_i^{0.50}|}{2} = 4 \neq |E|$

- Selon les résultats obtenus de la table 3.2, la meilleure classification des 6 bases de données ayant une valeur *goodness* maximale est *class*($D$,*sim*,0.45)={{$D_2$, $D_4$, $D_6$},{$D_1$, $D_3$, $D_5$}}




Pour $\delta$=0.79 :

{$D_2$,$D_4$}, {$D_1$}, {$D_3$}, {$D_5$}, {$D_6$}

Pour $\delta$=0.56 :

{$D_2$,$D_4$}, {$D_1$,$D_3$}, {$D_5$}, {$D_6$}

Pour $\delta$=0.50:

{$D_2$,$D_4$}, {$D_1$,$D_3$,$D_5$}, {$D_6$}

**Pour $\delta$=0.45 :**

**{$D_2$,$D_4$,$D_6$}, {$D_1$,$D_3$,$D_5$}**

Pour $\delta$=0.40 :

{$D_1$,$D_2$,$D_3$,$D_4$,$D_5$,$D_6$}

**FIGURE 3.3 -** LES CLASSIFICATIONS CANDIDATES GENEREES A PARTIR DE DE LA TABLE DE SIMILARITE

**TABLE 3.1.**

Dans la section suivante, nous décrivons les structures de données et les algorithmes utilisés dans l'approche de classification proposée.

## 3.2 Algorithmes et structures de données :

Le problème de génération d'une classification *class*(*D,sim,δ*) à un niveau de similarité *δ* est relié principalement aux sous-problèmes suivants :

1) Trouver la liste des arêtes ayant un poids égal à *δ*, notée $E_\delta$ et l'ajouter au graphe *G*.
2) Déterminer les composantes connexes de *G* et vérifier si chacune d'elle forme une clique.

Pour résoudre le premier problème, nous devons utiliser une structure de données de type dictionnaire pour une recherche efficace des poids dupliqués. Dans notre approche, nous devons lister les *m* listes $E_\delta$ dans l'ordre décroissant de leurs poids *δ*. Donc, nous avons utilisé un arbre binaire de recherche balancé, les arbres Rouge-Noir [Cormen 2009], pour implémenter une structure de données de type dictionnaire ordonné.

### 3.2.1 Structures de données :

Soit T un objet de type arbre balancé et T.*root* le nœud racine de l'arbre. Chaque objet de type nœud dans T contient principalement deux pointeurs (fils gauche et fils droit) et deux champs *weight* et *index* qui représentent respectivement la valeur du poids *δ* et l'indice de la première arête dans la liste $E_\delta$ ayant le poids *δ*. Les arêtes de même poids *δ* sont liées dans une table contigüe V[1..($n^2$-*n*)/2]. Les détails d'implémentation sont donnés dans les paragraphes qui suivent.

Par exemple, soit *x* un objet de type nœud dans T, alors V[x.*index*] contient l'indice de la seconde arête dont le poids est égale à x.*weight* et V[V[x.*index*]] contient l'indice de la troisième arête et ainsi de suite, jusqu'à ce qu'on trouve une valeur négative (-1) , qui représente la fin de la liste $E_\delta$.

Dans la procédure suivante, nous présentons l'algorithme utilisé pour construire les listes d'arête ordonnées $E_\delta$.

**Procédure *Build_Edge_Tree*(*n,* W)**

*Entrées* : *n* : nombre de bases de données;
W[1..($n^2$-*n*)/2]: la table de tous les poids des arêtes ($D_i$,$D_j$) pour *i*=1 à *n*-1, *j*=*i*+1 à *n*
*Sorties* : T : l'arbre des listes d'arête $E_\delta$.;
V[1..($n^2$-*n*)/2] : le table d'indices utilisée pour lier les arêtes de même poids;
*dist_inter* : la distance inter-cluster de la 1ère classification {{$D_1$},{$D_2$},…,{$D_n$}}

01. *dist_inter*=0;
02. T.*root*=null
03. {// *k* est l'indice de l'arête courante//}
04. **Pour** *k*=1 **à** ($n^2$-*n*)/2 **faire**
05. $\delta$ =W[*k*] ;
06. *dist_inter* = *dist_inter* + (1– $\delta$);
07. V[*k*]= -1;
08. *x*=***Tree_Search***(T.*root*, $\delta$); {//Rechercher un nœud dont le poids est égal à $\delta$//}
09. **Si** *x*≠ null et *x.weight*=$\delta$ **Alors**
10. V[*k*]=*x.index;*
11 *x.index=k;* {//Ajouter la *k*-ème arête à la liste $E_\delta$//}
12. **Sinon**
13. Allouer un nouveau nœud *y*;
14. *y.weight*=$\delta$*;*
15. *y.index=k;*
16. ***Tree_Insert***(T,*y*){//Insérer le nœud y dans l'arbre T *//}*
17. ***Tree_Rebalance***(T) {//Réarranger l'arbre après l'opération d'insertion//}
18. **Fin Si**
19. *k*=*k*+1*;*
20. **Fin boucle Pour**
21. Renvoyer (T, V[0..$C_n^2$-1], *dist_inter*);

**Fin procédure**

**FIGURE 3.4 -** ALGORITHME DE CONSTRUCTION DE L'ARBRE DE LISTES D'ARETE ORDONNEES EΔ.

**Discussion et analyse :**

Nous analysons les lignes de la procédure ***Build_Edge_Tree*** tout en déterminant la complexité temporelle de chaque ligne de code.

- Initialement en ligne 2, l'arbre T est vide. Ensuite, pour chaque poids $\delta$=W[$k$] correspondant à une paire de base de données ($D_i$,$D_j$) en ligne 4, une recherche dans l'arbre binaire balancé est exécutée en ligne 8 pour savoir si $\delta$ existe dans T.

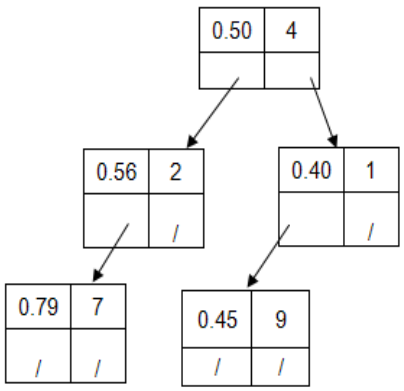
- S'il existe un nœud $x$ dans T tel que x.*weight* = $\delta$ alors l'arête courante (e.i., la $k$-ème arête) est insérée en tête de la liste $E_\delta$ en lignes 10-11. Sinon, un nouveau nœud, noté $y$, est alloué et inséré dans T en lignes 13-16 tel que l'attribut poids y.*weight* est initialisé à $\delta$ et l'attribut indice y.*index* est initialisé à $k$. La procédure auxiliaire *Tree_Rebalance* est appelée pour réarranger l'arbre après l'opération d'insertion afin de maintenir la hauteur de l'arbre proportionnelle au logarithme du nombre de nœuds dans T, en l'occurrence $O(log_2(m))$ [Cormen 2009].

- L'implémentation de *Tree_Rebalance* dépend du champ d'information supplémentaire (bit couleur Rouge/Noir, hauteur, etc.) et des opérations de rotation exécutées pour rééquilibrer et corriger l'arbre lorsque certaines de ses propriétés (telle que la couleur et/ou hauteur des nœuds) sont violées [Cormen 2009].
  Si $m \in [1,(n^2-n)/2]$ est le nombre de nœuds dans T, alors la recherche, l'insertion et les opérations de rotation prennent au pire des cas $O(log_2(m))$.

- Il existe $(n^2-n)/2$ valeurs de similarité entre les $n$ bases de données. Pour chaque valeur de similarité $\delta$, un nouveau nœud est inséré en T s'il n'existe aucun nœud dans T contenant le poids $\delta$. Donc, la complexité temporelle nécessaire pour construire T est estimée à $O(n^2 log_2(m))$ au pire des cas.

**Exemple :**

L'arbre T et la table d'indices V correspondants à la table de similarité table 3.1 sont illustrés dans la figure 3.5




| 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 |

**FIGURE 3.5** -L'ARBRE BINAIRE DE RECHERCHE T ET LA TABLE D'INDICES V (LA COLONNE « INDEX ») DE L'EXEMPLE DANS **TABLE 3.1**

La table située à la droite de la figure 3.5 représente la table d'indice V[1..($n^2$-$n$)/2] dans laquelle les arêtes avec le même poids sont liées. Seule la colonne « index » existe réellement en mémoire centrale. Les autres colonnes sont juste dans la table pour des buts d'illustration.

**Propriété 4 :**

Soit ($D_i$,$D_j$) une paire de nœuds (correspondant au $k$-ème arête pour $i$=1 à $n$-1, $j$=$i$+1 à $n$ et $k$=1 à ($n^2$-$n$)/2. Alors, l'indice ligne $i$ et l'indice colonne $j$ de la $k$-ème arête sont calculés comme suit :

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

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

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

Tel que l'opérateur ⌈ ⌉ renvoie le plus petit entier supérieur ou égale à l'argument et $C_n^2$ = ($n^2$-$n$)/2 est le nombre total d'arêtes dans un graphe complet de $n$ nœuds.

**Démonstration :**

- Soit M la table de similarité triangulaire de taille $n \times n$.
- Il y a en tout $C_n^2$ valeurs de similarité dans M, lesquelles sont associées avec $C_n^2$ arêtes indexées de 1 à $C_n^2$ :
- La 1 ère ligne de M est associée avec $n$-1 arêtes ($D_1$,$D_2$),($D_1$,$D_3$),…($D_1$,$D_n$), la seconde ligne avec $n$-2 arêtes ($D_2$,$D_3$),($D_2$,$D_4$),…($D_2$,$D_n$) et ainsi de suite jusqu'à la ($n$-1)-ème dernière ligne, laquelle est associée avec une arête ($D_{n-1}$,$D_n$). Ainsi, les dernières $h$ lignes de M sont associées avec $h(h+1)/2$ arêtes.
- Pour un indice donné $k$, il y a $C_n^2$-$k$+1 arêtes en comptant de la $k$-ème arête à la dernière arête, e.i., la $C_n^2$-ème arête.
- Pour trouver l'indice ligne $i$ de la $k$-ème arête, nous devons soustraire $h$ de $n$, tel que $h$ est l'entier qui vérifie: $h(h+1)/2 \geq C_n^2 - k + 1$.
- En résolvant l'inégalité on trouve :$(h+1/2)^2 \geq (8(C_n^2 - k + 1) + 1)/4$ et ça nous donne à la fin $h \geq (\sqrt{8 \times (C_n^2 - k + 1) + 1} - 1)/2$
- Donc, $i = n - h$ tel que $h = \left\lceil (\sqrt{8 \times (C_n^2 - k + 1) + 1} - 1)/2 \right\rceil$. Et on sait que la première colonne dans la ligne $i$ se trouve à l'indice $i$+1. Donc, pour trouver l'indice $j$ de la $k$-ème arête, nous devons ajouter le terme $(h(h+1)/2 - (C_n^2 - k + 1))$ au terme ($i$+1).

### 3.2.2 Algorithme de clustering des multi-bases de données :

Pour résoudre le second problème, qui est la détermination et la maintenance des composantes connexes de $G$, nous utilisons une structure de données appelée *forêt d'ensembles disjoints* [Cormen 2009]**.** Cette structure de données maintient une collection de $n$ arbres disjoints, où chaque arbre représente une composante connexe dans le graphe $G$=($D$,$E$,$W$). Le nœud racine de chaque arbre est utilisé comme nœud représentatif pour identifier une composante connexe unique. Chaque nœud dans la forêt d'ensembles disjoints, noté $x_i$ ($i$=1 to $n$), possède deux attributs principaux *size* et *parent*, lesquels représentent respectivement la taille du sous arbre enraciné au nœud $x_i$ et un pointeur vers son nœud parent.

- Dans cette approche, au lieu d'utiliser des pointeurs, nous utiliserons des entiers pour lier les nœuds fils à leurs nœuds parents. Par exemple, si $x_j$ est le nœud parent de $x_i$ , alors on met $x_i.parent=j$.

- La structure de données possède deux opérations principales décrites comme suit :

➢ ***union*($x_i$, $x_j$) :** combine les deux arbres identifiés par les nœuds racines $x_i$ et $x_j$. Autrement, elle lie la racine du plus petit arbre à celle du plus grand arbre. Après une opération *union*, la taille de l'arbre résultant est égale à la somme des tailles des arbres en entrée. La nouvelle taille *size* est mise à jour au niveau du nœud racine du plus grand arbre.

➢ ***findset*($x_i$) :** renvoie le nœud racine de l'arbre de $x_i$, c.à.d, cette procédure parcourt les nœuds de l'arbre en commençant du nœud parent de $x_i$, jusqu'à atteindre le nœud racine. Pour chaque arête incidente au *i*-ème nœud, *findset* est appelée sur le nœud $x_i$ pour retrouver la composante connexe à laquelle le *i*-ème nœud appartient.

- Dans cette thèse, nous avons légèrement modifié l'implémentation de la procédure *findset* afin d'identifier des *cliques* (e.i., sous-graphe complets) dans $G=(D,E,W)$. Si $x_i$ ou son nœud parent (noté $x_j$) est un nœud racine, alors *findset*($x_i$) renvoie le nœud racine d'entre eux. Sinon, $x_i$ est lié au parent de $x_j$ et *findset*($x_i$) renvoie $x_j.parent$ même si ce dernier n'est pas un nœud racine. Ainsi, pour chaque arête incidente au *i*-ème nœud, *findset*($x_i$) compresse le chemin entre $x_i$ et la racine quand $x_i$ ou $x_i.parent$ ne représentent pas la racine. Donc, si l'arbre de $x_i$ représente une composante connexe de *k* nœuds dans *G* et l'appel à *findset*($x_i$) ne renvoie pas un nœud racine, alors il devient clair qu'il y a au moins une arête manquante entre le *i*-ème nœud et les *k*-1 nœuds restants de la même composante. Par conséquent, la composante connexe, contenant le *i*-ème nœud n'est pas une clique dans *G*.

- Dans ce qui suit, nous présentons notre algorithme de clustering qui prend en entrée les objets renvoyés par la procédure *BuildEdgeTree*. En outre, nous présentons aussi les procédures *findset*, *union* et *makeset*. La procédure *makeset* permet juste d'initialiser les nœuds de la forêt d'ensembles disjoints.

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

*Entrée* : *n* : le nombre de bases de données;
$D=\{D_1,D_2,\ldots,D_n\}$ : l'ensemble des base de données;
T : l'arbre binaire balancé de toutes les listes d'arête;
V[1..$(n^2-n)/2$] :la table d'indice utilisée pour lier les arêtes de même poids;
*dist_inter* : la distance inter-cluster de $\{\{D_1\},\{D_2\},\ldots,\{D_n\}\}$ ;

*Sortie* : *class*(*D,sim,δ*) : la meilleure classification ;
01. A[1..*n*] : le tableau référençant les composantes connexes de *G*=(*D,E,W*);
02. *nb_component* =*n*; //nombre de composantes connexes initiales.
03. *sim_intra*= 0; //la similarité intra-cluster
04. *goodness_max*= |*dist_inter*+ *sim_intra* – *n*|;//la valeur *goodness* maximale
05. *nb_edges*=0;//nombre d'arêtes examinées
06. *nb_clique_edges*=0;//nombre d'arêtes requises pour produire des cliques dans *G ;*
07. *best_class_version*=1; //Version de la meilleure classification
*class_version*=1;//Version de la classification courante;
08. ***makeset***(A);
09. **Pour chaque nœud** *e* ∈T pris dans l'ordre décroissant du champ *weight* **faire**
10. Incrémenter *class_version* par 1;
11. *k*= *e.index;*
12. *δ*= *e.weight;*
13. **Tant que** *k*≠ – 1 **faire**
14. **Soit (**$D_i,D_j$) la *k*-ème arête;
15. incrémenter *nb_edges* par 1;
16. *dist_inter* = *dist_inter* – (1– *δ*); *sim_intra* = *sim_intra* + *δ*;
17. xi=***findset***(A[*i*], *best_class_version*, *class_version*);
18. xj=***findset***(A[*j*], *best_class_version*, *class_version*);
19. **Si (**xi.*parent*= – 1 et xj.*parent*= – 1 et xi≠ xj) **alors**
20. *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. **Fin Si**
24. *k*=V[*k*];
25. **File boucle Tant que**
26. **Si** *nb_edges*≠ *nb_clique_edges* **alors**
27. Ecrire **"***class*(*D,sim,δ*) n'est pas une classification complète**"**
28. **Sinon**
29. *temp*=|*sim_intra* +*dist_inter* – *nb_component*|;
30. **Si** *temp*>*goodness_max* **alors**
31. *goodness_max*=*temp*;
32. *best_class_version*=*class_version*;
33. **Fin Si**
34. **Fin Si**
35. **Fin boucle Pour**

36. **Pour** *i*=1 **à** *n* **faire**
37. **Si** A[*i*].*version*≤*best_class_version* **alors**
38. **Ecrire "**A[*i*].*parent* est l'indice du cluster de la *i*-ème base de données"
39. **Sinon Ecrire "**A[*i*].*parent_temp* est l'indice du cluster de la *i*-ème base de données"
40. **Fin Si**
41. **Fin boucle Pour**

**Fin Algorithme**

**Procédure *makset*(**A**)**

*Entrée*: A[1..*n*]: le tableau représentant les *n* nœuds de la *forêt d'ensembles disjoints*.

01. **Pour** *i*=1 **à** *n* **faire**
02. A[*i*]= allouer un nouveau nœud objet xi;
03. xi.*parent*= – 1;
04. xi.*size*=1;
05. xi.*parent_temp*= – 1;
06. xi.*version*=1;
07. **Fin pour;**

**Fin procédure**

**Procédure *union*(**$x_i$, $x_j$,*class_version,best_class_version***)**

*Entrée* : xi, xj: deux composante disjointe à fusionner
*class_version* : la version de la classification courante
*best_class_version* : la version de la meilleure classification

01.**Si** xi.*size*>xj.*size* **alors**
02. **Si** xj.*version* ≤ *best_class_version* **alors** xj.*parent_temp*= xj.*parent;* **Fin Si**
03. xj.*parent*=i;
04. xj.*version*=*class_version,//* Mettre à jour la version courante du parent de xj
05. xi.*size*=xi.*size*+xj.*size*;
06. **Sinon**
07. **Si** xi.*version*≤*best_class_version* **alors** xi.*parent_temp*=xi.*parent;* **Fin Si**
08. xi.*parent*=j;
09. xi.*version*=*class_version,//* Mettre à jour la version courante du parent de xi
10. xj.*size* =xj.*size* +xi.*size*;
11. **Fin Si**

**Fin procédure**

**Procédure *findset*(**$x_i$, *class_version,best_class_version*)**

*Entrée*: A[1..*n*]: le tableau représentant les *n* nœuds de la *forêt d'ensembles disjoints*.
xi : le *i*-ème noeud objet
*class_version* : la version de la classification courante
*best_class_version* : la version de la meilleure classification

*Sortie*: Le noeud racine ou le nœud parent de xi ou le nœud grand-père de xi

```
01. Si xi.parent = – 1 alors renvoyer xi;
02. Sinon Si A[xi.parent].parent = – 1 alors renvoyer A[xi.parent];
03. Sinon
04.     Si xi.version≤best_class_version alors xi.parent_temp=xi.parent; Fin Si;
05.     xi.parent= A[xi.parent].parent;
06.     xi.version=class_version,//mettre à jour la version du nœud parent courant xi.parent;
07.     renvoyer A[xi.parent];
08. Fin Si;
Fin procédure
```

**FIGURE 3.6 -** ALGORITHMES PROPOSES POUR LE CLUSTERING DES MULTI-BASES DE DONNEES.

**Discussion et analyse :**

Nous expliquons les étapes de l'algorithme *BestDatabaseClustering* tout en déterminant sa complexité temporelle et spatiale.

**a) Etape d'initialisation dans *BestDatabaseClustering*:**

- La distance inter-cluster (nommée *dist_inter*) entre les *n* composantes a déjà été calculée dans le corps de la procédure *BuildEdgeTree* puis passée comme entée à l'algorithme *BestDatabaseClustering*.

- Initialement, le graphe $G=(D,E,W)$ possède *n* composantes disjointes, e.i., *n* singletons. A cette étape, on obtient une classification triviale de *n* clusters $class(D,sim)=\{\{D_1\},\{D_2\},\ldots,\{D_n\}\}$. Les lignes 1-7 permettent d'initialiser les variables utilisées dans l'algorithme.

- Au début, la similarité intra-cluster (nommée *sim_intra*) est nulle, car il y a une seule base de données dans chaque cluster. La variable *goodness_max* est utilisée pour maintenir la valeur maximale de la mesure *goodness* trouvée à présent.

- En ligne 8, la procédure *makeset* initialise la structure de données en créant *n* arbres, chacun possède un seul nœud $x_i$ qui est la racine de l'arbre, e.i., $x_i.size$=1 pour $i$=1 à $n$.

- Pour accéder aux $n$ nœuds, nous utilisons le tableau A[1..$n$] tel que A[$i$] contient le nœud $x_i$ représentant le $i$-ème nœud dans $G$. La complexité temporelle de la ligne 8 dans *BestDatabaseClustering* est de $O(n)$, car la procédure *makeset* permet d'allouer et d'initialiser $n$ nœuds en mémoire centrale.

**b) Etape de parcours des listes d'arête $E_\delta$ dans *BestDatabaseClustering*:**

- En ligne 9, la boucle *Pour* examine chaque nœud $e$ de l'arbre T dans l'ordre décroissant du poids e.*weight*. Il existe $m$ nœuds dans T tel que $1 \leq m \leq (n^2-n)/2$ et chaque nœud représente une liste d'arête $E_\delta$. Donc, parcourir tous les $m$ nœuds prend $O(m)$ unité de temps en utilisant un parcours récursif infixé de l'arbre binaire T : D'abord parcourir le sous arbre gauche à partir de la racine $e$, puis remonter à la racine $e$ et à partir de la explorer le sous arbre droit. La complexité temporelle d'une procédure de parcours infixé d'un arbre binaire de recherche de taille $m$ est de l'ordre $O(m)$.

**b-1) Etape de parcours des arêtes de même poids de la liste courante $E_\delta$ dans *BestDatabaseClustering*** :

- En ligne 13, la boucle *Tant-que* examine la liste d'arêtes courante $E_\delta$ où $\delta$ = e.*weight.* Les arêtes sont parcourues en commençant à partir de la première arête, dont l'indice est e.*index*, jusqu'à atteindre une valeur d'indice négative (-1), qui représente la fin de la liste d'arête. En ligne 14, les équations (7) et (8) sont utilisées pour trouver les deux nœuds ($D_i$,$D_j$) correspondant à la $k$-ème arête (e.i., l'arête courante).

- En lignes 17-18, la procédure *findset* est appelée pour chaque nœud de la $k$-ème arête ($D_i$, $D_j$) pour trouver les composantes à laquelle $D_i$ and $D_j$ appartiennent. La complexité temporelle de chacune des lignes 17-18 dans *BestDatabaseClustering* est constante, de l'ordre $O(1)$, car la procédure *findset* permet de retourner le nœud parent du nœud paramètre en entrée. La procédure *findset* permet aussi de faire pointer le nœud paramètre à son nœud grand parent afin de compresser le chemin entre chaque nœud avec le nœud racine de l'arbre.

- Soit $x_i$ et $x_j$ les nœuds retournés par *findset.* Si $x_i$ et $x_j$ sont deux nœuds racine différents en ligne 19, alors la procédure *union* (en ligne 21) va fusionner les arbres correspondants en pointant la racine du petit arbre à la racine du grand arbre. Sinon, soit la classification courante n'est pas complète, soit les deux nœuds $D_i$ and $D_j$ appartiennent déjà à la même composante.

- En ligne 20, on met à jour *nb_clique_edges*, qui est le nombre d'arêtes requis pour que chaque composante connexe de *G* soit une clique (sous-graphe complet). Après chaque opération *union*, le nombre de composantes dans le graphe *nb_component* est décrémenté.

- En ligne 24, on sélectionne l'arête suivante à traiter dans la liste $E_\delta$ à partir du tableau V[1..($n^2$-$n$)/2].

**b-2) Etape d'évaluation de la classification courante dans *BestDatabaseClustering***:

- En ligne 26, la classification courante est vérifiée si elle est complète ou pas en utilisant l'équation (6). Si le nombre d'arêtes examinées à présent, *nb_edges*, est égal à *nb_clique_edges*, alors la classification est complète et l'algorithme continue à la ligne 29. Sinon la classification courante est ignorée.

- En lignes 29-32, la classification courante *class*(*D,sim,*$\delta$) est évaluée en utilisant la mesure *goodness* (voir définition 5). Cette mesure se base sur la maximisation de la similarité intra-cluster et la distance inter-cluster. Tant que *goodness*(*class,* $\delta$) $\geq$ *goodness_max*, *class*(*D,sim,*$\delta$) est la meilleure classification générée à présent et la valeur de *goodness_max* est mise à jour en ligne 31.

**c) Etape de traçage et d'affichage de la meilleure classification dans *BestDatabaseClustering* :**

- Au cours de l'exécution de notre algorithme, nous avons besoin de garder une trace de la meilleure classification générée à présent au fur et à mesure que la *forêt d'ensembles disjoints* A[1..*n*] est mise à jour. Pour implémenter une telle structure, des champs additionnels (*version* et *parent_temp*) sont associés à chaque nœud A[*i*]. Ces champs sont requis pour maintenir la version des clusters au moment où la meilleure classification a été générée.

- Soit *x* un nœud dans la *forêt d'ensembles disjoints*. Le champ x.*parent_temp* est utilisé pour garder une copie du nœud parent de *x* trouvé dans la meilleure classification générée à présent. Tandis que le champ x.*version* est utilisé pour déterminer la classification à laquelle correspond le nœud parent de *x*.

- Dans notre algorithme, chaque classification *class*(*D*,*sim*, $\delta$) est identifiée par sa son numéro de version *class_version*. Soit *best_class_version* le numéro de version de la meilleure classification. Avant chaque opération de mise à jour dans la procédure *union*/*findset*, chaque nœud *x* est vérifié pour voir si son nœud parent courant fait partie de la meilleure classification trouvée à présent. Donc, si *x.version*≤*best_class_version*, alors l'indice du parent de *x* est stocké dans le champ *x.parent_temp* avant l'opération de mise à jour afin de garder sa trace au cas où on voudrait trouver sa valeur plus tard.

- En lignes 36-41, on affiche la meilleure classification des *n* bases de données à partir de la *forêt d'ensembles disjoints* A[1..*n*]. Ainsi, l'indice du parent de chaque nœud $x_i$ $(1 \leq i \leq n)$ est affiché au moment où la meilleure classification a été générée. Donc, si $x_i$.*version*≤*best_class_version*, alors $x_i$.*parent* est renvoyé comme étant le cluster contenant la *i*-ème base de donnée. Sinon, c'est $x_i$.*parent_temp* qui est renvoyé à sa place.

**d) Analyse de la complexité temporelle globale de *BestDatabaseClustering*:**

- L'algorithme proposé *BestDatabaseClustering* [Miloudi et al. 2018] s'achève une fois toutes les *m* listes d'arêtes $E_\delta$ examinées et les *m* classifications candidates possibles générées. La taille moyenne d'une seule liste d'arête est $\frac{n^2}{m}$ arêtes. Ainsi, explorer toutes les *m* listes d'arêtes prend $O(\frac{n^2}{m} \times m)$=$O(n^2)$ temps au pire des cas, tel que *n* est le nombre de bases de données à classer.

- Donc, l'algorithme proposé trouve la meilleure classification de l'ensemble des *n* bases de données en un temps estimé à $O(n^2)$ au pire des cas. Notre algorithme est optimal comparé à *BestDatabasePartition* [Adhikari et al. 2010a], lequel prend $O(m \times n^2)$ pour trouver la meilleure partition des *n* multi-bases de données.

**e) Analyse de la complexité spatiale globale de *BestDatabaseClustering* :**

- La complexité spatiale de notre algorithme *BestDatabaseClustering* [Miloudi et al. 2018] est estimée comme suit. Pour stocker les $m$ nœuds de l'arbre T en mémoire, nous avons besoin de $m$ unités de (2×pointeurs+1×réel+1×entier). Nous devons aussi ajouter l'espace requis pour stocker le tableau d'indice V, donc, $(n^2-n)/2$ unités d'entier.

- De plus, la structure de donnée des *ensembles de forets disjoints* a besoin de $n$ unités de (4 entiers). Donc, en ajoutant l'espace requis pour stocker les $(n^2-n)/2$ valeurs de similarité ($(n^2-n)/2$ réels), l'espace mémoire utilisé par notre algorithme est en moyene $(6n^2+10n+28m)$ octets dans le cas d'un compilateur qui représente un pointeur avec 8 octets, un entier avec 4 octets et un nombre réel avec 8 octets respectivement.

- En parallèle, *BestDatabasePartition* [Adhikari et al. 2010a] a besoin de $n^2$ unités de réels pour stocker la table de similarité. En supposant qu'il utilise un arbre binaire de recherche balancé pour trier toutes les valeurs distinctes de similarité, alors $m$ unités de (2×pointeurs+réel) seront requises pour stocker tous les nœuds de l'arbre. Chaque classification candidate est stockée séparément. Ainsi, $mn$ unités d'entier sont requises pour stocker toutes les $m$ classification candidates. Donc, l'espace mémoire requis pour *BestDatabasePartition* [Adhikari et al. 2010a] est en moyenne $(8n^2+20m+4mn)$ octets en utilisant le même compilateur précèdent. Pour de grande valeurs de $n$ et $m$, nous pouvons remarquer que notre algorithme consomme moins d'espace mémoire que *BestDatabasePartition* [Adhikari et al. 2010a].

# 4 Amélioration de la précision de la mesure de similarité

La précision d'un algorithme de clustering est fortement déterminée par le calcul de la similarité entre les bases de données. Dans le travail proposé par [Adhikari et al. 2010a], plus le nombre d'itemsets fréquents partagés entre les bases de données est grand, plus les clusters de bases de données formés sont cohésifs et pertinents. Toutefois, les algorithmes existants négligent les itemsets locaux non fréquents, dont le support est inférieur au seuil minimum, mais qui sont d'une grande importance à la de fouille multi-bases de données (FMBD). En effet, ces itemsets locaux non fréquents peuvent grandement contribuer à la synthèse des itemsets globaux d'un cluster de bases de données et ainsi améliorer la qualité des motifs trouvés.

Dans les sections suivantes, nous proposons une mesure de similarité basée sur l'estimation des supports des itemsets non fréquents dans les multi-bases. Ainsi, seules les bases de données qui améliorent la synthèse des itemsets globaux sont mises dans le même cluster. Pour cela, nous utilisons le modèle proposé par [Ramkumar and Srinivasan 2008] avec le facteur de correction défini par [Srinivasan and Ramkumar 2009] pour synthétiser le support global d'un itemset dans les bases sur lesquelles on veut calculer la similarité.

## 4.1 Méthode et métriques proposées :

La mesure de similarité $sim_3$ proposée par [Adhikari et al. 2010a] présente une certaine insuffisance. Le numérateur dans $sim_3(D_i,D_j)$ prend en compte seulement les itemsets fréquents qui sont partagés entre deux bases de données $D_i$ et $D_j$ et ignore les motifs supportés par uniquement l'une des deux bases de données. Lorsque un motif $X$ n'est pas fréquent dans $D_i$, e.i, $X \notin FIS(D_i,\alpha)$, cela ne signifie pas que $X$ n'est pas du tout présent dans $D_i$. Il peut être présent avec un support dont la valeur est inférieur mais proche du seuil du support minimum α. Ainsi, $X$ peut être fréquent lorsque les deux bases de données sont intégrées et une fouille de données est appliquée. Donc, pour faire participer la contribution de $X$, nous devons estimer le support de $X$ dans l'ensemble $\{D_i \cup D_j\}$ dans le cas où $supp(X,D_i)<\alpha$ ou $supp(X,D_j)<\alpha$. Pour cela, nous avons utilisé le modèle de synthèse proposé dans [Ramkumar and Srinivasan 2008] comme suit.

**Définition 8 :**

Le support estimé de $X$ dans l'union $\{D_i \cup D_j\}$ tel que défini par [Ramkumar and Srinivasan 2008] est donné comme suit :

$$supp_s(X, D_i \cup D_j) = \frac{supp(X,D_i)\times|D_i| + supp(X,D_j)\times|D_j|}{|D_i|+|D_j|} \quad (9)$$

Tel que $supp(X,D_i)$ est le support local de $X$ dans $D_i$ et $|D_i|$ est le nombre de transactions dans $D_i$.

En utilisant la formule (9), nous proposons la mesure de similarité suivante.

**Définition 9 :**

Soit $D_i$ et $D_j$ deux bases de données transactionnelles et α la valeur du support minimum. La mesure $sim(D_i,D_j)$ définie par [Miloudi et al. 2016a] est présentée comme suit :

$$sim\ (D_i, D_j, \alpha) = \frac{\Sigma_{X\in(FIS(D_i,\alpha)\cup FIS(D_j,\alpha))}\ \phi\{X,D_i\cup D_j,\alpha\}}{\Sigma_{X\in(FIS(D_i,\alpha)\cup FIS(D_j,\alpha))}\ max\{\ supp(X,D_i)\ ,supp(X,D_j)\ \}} \quad (10)$$

Tel que,

$$\phi\{X, D_i \cup D_j\ , \alpha\} = \begin{cases} supp_s(X, D_i \cup D_j)\ , si\ supp_s(X, D_i \cup D_j) \geq \alpha \\ 0, sinon \end{cases} \quad (11)$$

La fonction *sim* calcule la similarité entre deux bases de données en utilisant les supports globaux synthétisés des itemsets fréquents. Le support global synthétisé d'un itemset fréquent représente une estimation de son support réel obtenu par une fouille de données appliquée à l'ensemble des bases intégrées. Donc, *sim* va contribuer à grouper correctement les bases de données à analyser afin d'augmenter la qualité des motifs découverts.

**Exemple :**

Soit $D_i$ et $D_j$ deux bases de données, chacune possède 10 transactions. Soit le seuil du support minimum α=0.2. Supposons que les itemsets fréquents obtenus à partir de chaque base de données sont comme suit :

$$FIS(D_i,\alpha)=\{supp(A,D_i)=0.5, supp(B,D_i)=0.4, supp(C,D_i)=0.2\}$$

$$FIS(D_j,\alpha)=\{supp(C,D_j)=0.3\}$$

Nous observons que les itemsets A et B ne sont pas fréquents dans $FIS(D_j,\alpha)$ car leurs supports sont inférieurs à $\alpha$=0.2, avec $supp(A,D_j)$=0.1 et $supp(B,D_j)$=0.1.

Calculons la similarité entre $D_i$ et $D_j$ en utilisant la mesure proposée par [Adhikari et al. 2010a] :

$$sim_3(D_i,D_j)= \frac{0+0+0.2}{0.5+0.4+0.3} = 0.167$$

Maintenant, en utilisant notre mesure de similarité, nous obtenons le résultat suivant :

$$sim(D_i,D_j)= \frac{\frac{5+0}{20}+\frac{4+0}{20}+\frac{3+2}{20}}{0.5+0.4+0.3} = \frac{0.25+0.20+0.25}{0.5+0.4+0.3} = 0.58$$

La mesure de similarité proposée par [Adhikari et al. 2010a] n'a pas pris en compte les supports des itemsets A et B dans $D_j$ car $supp(A,D_j)< \alpha$ et $supp(B,D_j)<\alpha$. Par conséquent, $sim_3(D_i,D_j)$ retourne une valeur très petite. Cependant, les itemsets A et B sont fréquents lorsque les deux bases de données $D_i$ et $D_j$ sont intégrées et une fouille de donnée est appliquée sur l'union des deux bases $\{D_i \cup D_j\}$.
En effet, les supports réels de A et B dans $\{D_i \cup D_j\}$ sont calculés comme suit :

$$supp(A, \{D_i \cup D_j\})=(5+1)/20=0.3 \text{ et } supp(B, \{D_i \cup D_j\})=(4+1)/20=0.25$$

Les supports globaux synthétisés de A et B dans $\{D_i \cup D_j\}$ sont proches des supports réels avec :

$$supp_s(A, \{D_i \cup D_j\})=(5+0)/20=0.25 \text{ et } supp_s(B, \{D_i \cup D_j\})=(4+0)/20=0.20$$

Cela signifie que, si les deux bases de données $D_i$ et $D_j$ sont mises dans un même cluster, alors la fouille de données de leur union va permettre de découvrir 3 itemsets fréquents globaux : A, B et C. Cependant, si la valeur de similarité entre $D_i$ et $D_j$ est très petite (comme dans le cas de l'utilisation de la mesure $sim_3(D_i,D_j)$ proposée par [Adhikari et al. 2010a], alors, les deux bases de données $D_i$ et $D_j$ seront considérées comme non pertinentes et ainsi des motifs utiles seront perdus. Ainsi, la mesure de similarité *sim* proposée par [Miloudi et al. 2016a] retourne des résultats plus précis.

De plus, *sim*, pourrait être améliorée encore plus en utilisant le facteur de correction *h* proposé par [Srinivasan and Ramkumar 2009]**.** En effet, nous pouvons estimer le support local de $X$ dans $D_i$ lorsque $supp(X,D_i)<\alpha$ comme suit.

**Définition 10 :** Le support local de $X$ dans $D_i$ pourrait être estimé comme suit :

$$supp(X, D_i) = \begin{cases} h \times \alpha \, , si \; supp(X, D_i) < \alpha \\ supp(X, D_i), sinon \end{cases} \quad (12)$$

Où $h \in [0,1]$ est le facteur de correction et α est le seuil du support minimum.

Le choix d'un facteur de correction *h* approprié est relié à la distribution des données dans la base. Plus les sous-ensembles de $X$ sont fréquents (e.i., $\forall\, x_i \subseteq X,\ supp(x_i, D_i) \geq \alpha$) avec un grand support, plus la probabilité de les voir apparaitre ensemble est grande et donc $supp(X,D_i)$ va être proche du minsupp $\alpha$. Ainsi, *h* doit être choisi proche de 1 dans ce cas. Dans l'absence d'une telle information, assigner *h*=0.5 reste un choix moyen approprié. Dans l'exemple précèdent, nous pouvons estimer le support local de A et B dans $D_j$ en appliquant un facteur de correction *h*=0.5 comme suit :

$$supp(A,D_j)=supp(B,D_j)=0.5\times0.2=0.1$$

Les supports locaux estimés sont dans ce cas exactement les supports locaux réels de A et B dans $D_j$. Donc, une nouvelle valeur de *sim* est obtenue comme suit :

$$sim(D_i,D_j)= \frac{\frac{5+1}{20}+\frac{4+1}{20}+\frac{3+2}{20}}{0.5+0.4+0.3} = 0.66$$

Comme nous pouvons le constater, $sim(D_i,D_j)$ indique que $D_i$ and $D_j$ sont pertinentes (avec une valeur de similarité supérieure à 50%) car tous les itemsets locaux de $D_i$ et $D_j$ seront fréquents après la fouille de l'union $\{D_i \cup D_j\}$. Par conséquent, il est plus approprié de mettre $D_i$ et $D_j$ dans le même cluster pour améliorer la qualité des itemsets globaux synthétisés.

Dans ce qui suit, nous proposons une autre version optimisée de $sim_3(D_i,D_j)$ [Adhikari et al. 2010a]. En effet, le fait de juste estimer le support d'un itemset non fréquent dans une base pourrait déjà améliorer la qualité de la classification générée.

**Définition 11.** En se basant sur le facteur de correction *h*, $sim_4$ pourrait être proposée comme suit :

$$sim_4(Di, Dj) = \frac{\sum_{x\in\left(FIS(D_i,\alpha)\cup FIS\left(D_j,\alpha\right)\right)} min\{\,\Phi(x,D_i)\,,\Phi\left(x,D_j\right)\,\}}{\sum_{x\in\left(FIS(D_i,\alpha)\cup FIS\left(D_j,\alpha\right)\right)} max\{\,\Phi(x,D_i)\,,\Phi\left(x,D_j\right)\,\}} \quad (13)$$

Tel que, $\Phi(X,D_i)$ est calculée par l'équation (12)

Donc, lorsque $X$ n'est pas fréquent dans $FIS(D_i,\alpha)$, son support local est estimé en utilisant le facteur de correction $h$. Ainsi, $sim_5(D_i,D_j)$ retournera des résultats plus précis.

**Exemple :**

Pour démontrer l'efficacité de la mesure de similarité $sim_4$, nous utilisons l'ensemble de base de données suivant. Soit $D_1$, $D_2$ et $D_3$ trois bases de données et leurs tailles (e.i nombre de transactions) sont 10,000, 30,000 et 10,000 respectivement.

Les itemsets fréquents des différentes bases sont donnés dans la table 3.3.

Avec α=0.2, l'itemset "*A C*" ne sera pas extrait de $D_3$ car son support est inférieur à α, avec $supp(A\ C, D_3)$=0.15. En outre, les itemsets "*A D*" et "*C E*" ne seront pas extraits de $D_1$, car $supp(A\ D, D_1)$=0.08 < α et $supp(C\ E, D_1)$=0.07< α

| **Bases de données** | **Taille (Nombre de Transactions)** | **Itemsets avec leurs supports** | | |
|---|---|---|---|---|
| | | ***A C*** | ***A D*** | ***C E*** |
| $D_1$ | 10,000 | 0.30 | **0.08** | **0.07** |
| $D_2$ | 30,000 | 0.40 | 0.35 | 0.40 |
| $D_3$ | 10,000 | **0.15** | 0.40 | 0.50 |

**Table 3.3 –**Les bases de données transactionnelles avec leurs itemesets correspondants sous α=0.2

En calculant la similarité entre les deux bases de données $D_1$ et $D_3$ avec $sim_3$ [Adhikari et al. 2010a] on obtient :

$$sim_3(D_1,D_3) = \frac{0+0+0}{0.3+0.4+0.5} = 0$$

La valeur obtenue est nulle car $sim_3$ prend en compte seulement les itemsets fréquents partagés entre deux bases de données. Puisque les itemsets "*A C*", "*A D*" and "*C E*" ne sont pas tous extraits de $D_1$ et $D_3$, le numérateur dans $sim_3$ est nul. Cependant, "*A C*", "*A D*" et "*C E*" existent vraiment dans les bases de données $D_1$ et $D_3$ et nous devrions estimer leurs supports pour avoir des résultats plus précis.

En utilisant le facteur de correction $h$=0.5, leurs supports estimatifs sont obtenus comme suit : $supp(A\ C, D_3)= supp(A\ D, D_1)=supp(C\ E, D_1)=h\times \alpha$=0.1, qui sont approximativement proches des valeurs réelles.

Donc, si nous calculons la similarité entre $D_1$ et $D_3$ tout en incluant les supports estimatifs nous obtiendrons la valeur suivante :

$$sim_4(D_1,D_3)= \frac{0.1+0.1+0.1}{0.3+0.4+0.5} = 0.25.$$

Les valeurs de similarité entre les bases de données restantes en utilisant $sim_3$ [Adhikari et al. 2010a] et $sim_4$ sont données dans la table 3.4

| **Bases de données** | $D_1$ | | $D_2$ | | $D_3$ | |
|---|---|---|---|---|---|---|
| | $sim_3$ | $sim_4$ | $sim_3$ | $sim_4$ | $sim_3$ | $sim_4$ |
| $D_1$ | 1 | 1 | 0.26 | 0.43 | 0 | 0.25 |
| $D_2$ | 0.26 | 0.43 | 1 | 1 | 0.57 | 0.65 |
| $D_3$ | 0 | 0.25 | 0.57 | 0.65 | 1 | 1 |

**Table 3.4 –**La table de similarité entre les bases de données dans **Table 3.3** avec les métriques $sim_3$ [Adhikari et al. 2010a] et $sim_4$ [Miloudi et al. 2016b]

Les résultats de classification des 3 bases de données en utilisant $sim_3$ [Adhikari et al. 2010a] et la mesure proposée $sim_4$ sont donnés dans la table 3.5.

Comme on peut le constater, $D_1$ n'appartient pas au cluster $\{D_2, D_3\}$ quand $sim_3$ est utilisée. Car $sim_3(D_1,D_3)$=0. Cependant, avec l'utilisation de $sim_4$, $D_1$ est ajoutée au cluster $\{D_2, D_3\}$ et la valeur *goodness* (voir définition 5) de la classification obtenue avec $sim_4$ est plus grande que celle obtenue avec $sim_3$. Ainsi, la mesure de similarité proposée $sim_4$ renvoie des résultats plus précis.

| $sim_3$ | | $sim_4$ | |
|---|---|---|---|
| $class(D,sim_3)$ | $goodness(D,sim_3)$ | $class(D,sim_4)$ | $goodness(D,sim_4)$ |
| $\{D_1\}, \{D_2, D_3\}$ | 0.31 | $\{D_1, D_2, D_3\}$ | 0.33 |

**Table 3.5 –**La meilleure classification des bases de données dans **Table 3.3** avec les métriques $sim_3$ [Adhikari et al. 2010a] et $sim_4$ [Miloudi et al. 2016b]

## 5 Conclusion

Dans ce chapitre nous avons présenté notre contribution pour l'amélioration des algorithmes de clustering des multi-sources de données. Dans le chapitre suivant nous allons présenter les expérimentations réalisées afin d'évaluer notre algorithme de clustering.

# Chapitre 4 -

## Expérimentations & Résultats

# 1 Etude expérimentale I

Pour valider la performance de notre algorithme de clustering, quelques expérimentations ont été menées sur des bases de données synthétiques et sur des tables de similarités pré-calculées. Les algorithmes étudiés ont été implémentés en JAVA Edition 6 et exécutés sur un PC CORE i7 cadencé à 2.60 GHz et ayant 6 GB de mémoire vive et 1 TB de capacité de disque dur. Les expérimentations réalisées sont présentées en trois parties I, II et III comme suit.

Afin d'évaluer la performance de notre algorithme, nous avons comparé son temps d'exécution avec celui de l'algorithme *BestDatabasePartition* [Adhikari et al. 2010a]. Ce choix est justifié par le faite que *BestDatabasePartition* [Adhikari et al. 2010a] a le plus court temps d'exécution comparé aux algorithmes proposés par [Wu et al. 2005] et [Li et al. 2009].

Nous avons mené les expérimentations sur deux ensembles de données synthétiques T10I4D100K et T40I10D100K disponibles sur http://fimi.ua.ac.be/data. Ces ensembles de données sont décrits dans la table 4.1. Nous notons par NT, ALT, AFI, NI and DB le nombre de transactions, la taille moyenne d'une transaction, la fréquence moyenne d'un item, le nombre d'items et le nom de base de données respectivement.

Pour la fouille multi-bases de données, chaque ensemble de données est divisé horizontalement en 10 et 20 bases de données. Les sous bases de données obtenues à partir de T10I4D100K et T40I10D100K sont référencées respectivement comme suit : $T_{1,1}$, $T_{1,2}$, …, $T_{1,n}$ et $T_{4,1}$, $T_{4,2}$, …, $T_{4,n}$ tel que $n$=10, 20

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

**Table 4.1 –** Ensembles de données synthétiques**.**

En variant la valeur du seuil du support minimum ($\alpha$), nous obtenons différents ensembles d'itemsets fréquents en utilisant l'algorithme FP-Growth [Han et al. 2000] comme présenté dans la table 4.2.

Nous notons par {FIS($T_{i,1}$, α), FIS($T_{i,2}$, α),…, FIS($T_{i,n}$, α)} les ensembles d'itemsets fréquents obtenus à partir des *n* multi-bases de données $T_{i,1}$, $T_{i,2}$, …, $T_{i,n}$ sous α tel que *i*=1,4. La figure 4.1 illustre l'étape de préparation des données expérimentales.

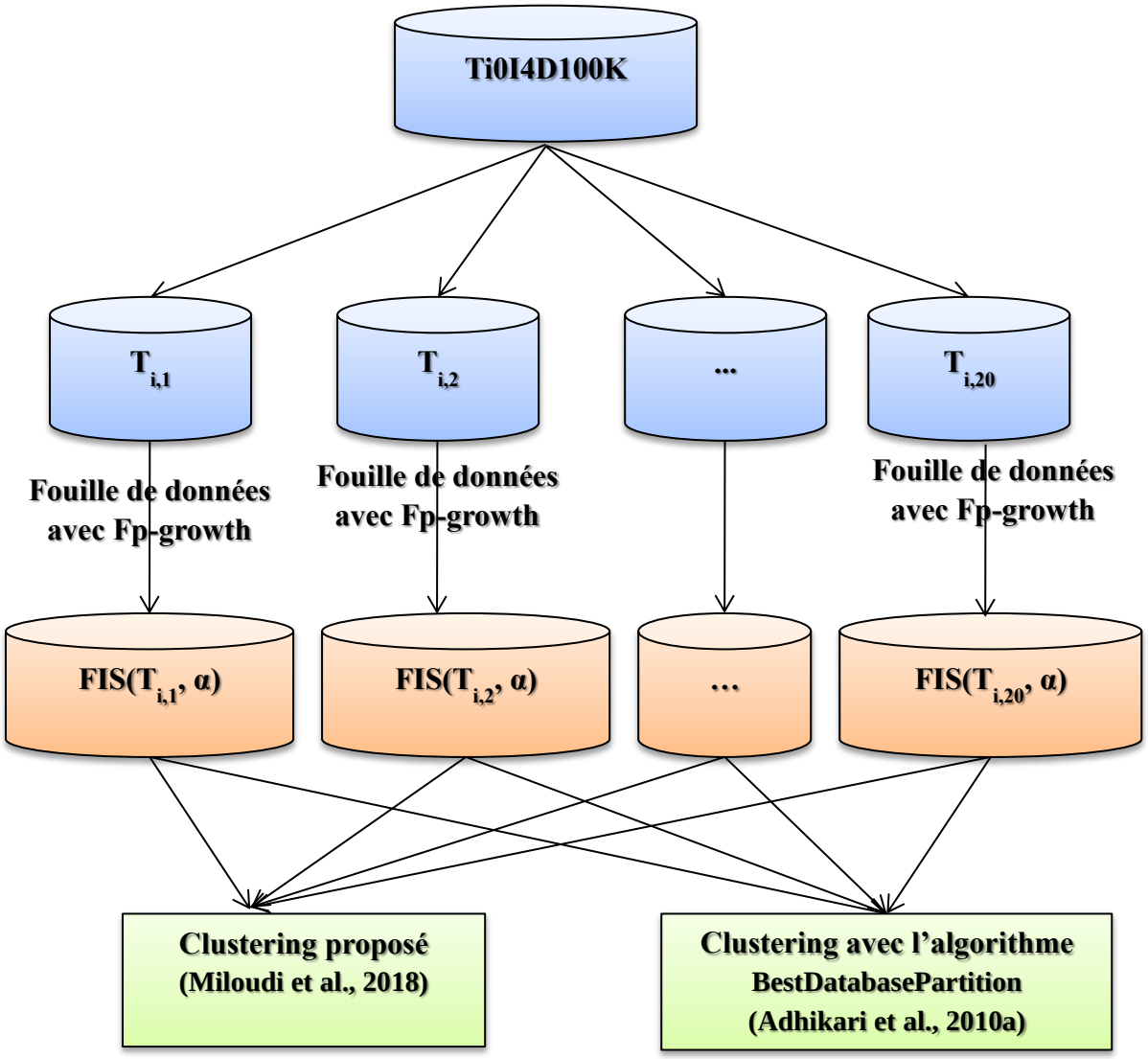



**Figure 4.1** - ETAPES DE PRETRAITEMENT DES DONNEES

| Dataset ($i$=1,4) | # de base de données ($n$) | # de transaction dans chaque base de données | Algorithme | Support Minimum ($\alpha$) | Ensembles d'itemsets fréquents |
|---|---|---|---|---|---|
| Ti | 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 4.2 –** Les ensembles d'itemsets fréquents obtenus à partir des multiples bases de données $T_{1,1}$, $T_{1,2}$, …, $T_{1,n}$ and $T_{4,1}$, $T_{4,2}$, …, $T_{4,n}$ pour $\alpha$=0.03 à 0.06 et $n$=10, 20

Une fois l'étape de prétraitement des données terminée, nous classifions les multi-bases de données en utilisant notre algorithme et *BestDatabasePartition* [Adhikari et al. 2010a]**.** Puisque il est difficile de mesurer la performance d'un algorithme exécuté par la machine virtuelle Java (JVM), à cause des optimisations faites par la JVM durant l'exécution du code source, chaque algorithme est exécuté 1000 fois sur les mêmes ensembles de bases de données afin d'obtenir le temps d'exécution moyen.

Dans la première étude expérimentale, nous analysons l'impact du seuil du support minimum minsupp ($\alpha$) sur le temps d'exécution du clustering. Pour cela, nous fixons le nombre de bases de données à $n$=10 et nous varions la valeur de minsupp($\alpha$) pour obtenir des ensembles d'itemsets fréquents différents.

Nous exécutons notre algorithme et *BestDatabasePartition* [Adhikari et al. 2010a] avec les même itemsets fréquents en entrée. Les résultats de classification et les temps d'exécution obtenus sont présentés en table 4.3 et la figure 4.2 respectivement.

## 2 Analyse des résultats expérimentaux de l'étude I

| Multiples bases de données | Nombre de classifications candidates (*m*) | Support Minimum (α) | Meilleure classification | *goodness* | Temps d'exécution (seconde) | |
|---|---|---|---|---|---|---|
| | | | | | *BestDatabase Partiton* | Algorithme proposé |
| $T_{1,1}$, ,…,$T_{1,10}$ | 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 |

**Table 4.3 –** Comparaison entre les résultats de classification obtenus par notre algorithme et *BestDatabasePartition* (Adhikari et al., 2010a) sur les multiple bases de données $T_{1,1}$ ,…, $T_{1,10}$ et $T_{4,1}$ ,…, $T_{4,10}$ pour α=0.03 à 0.06 et *n*=10.

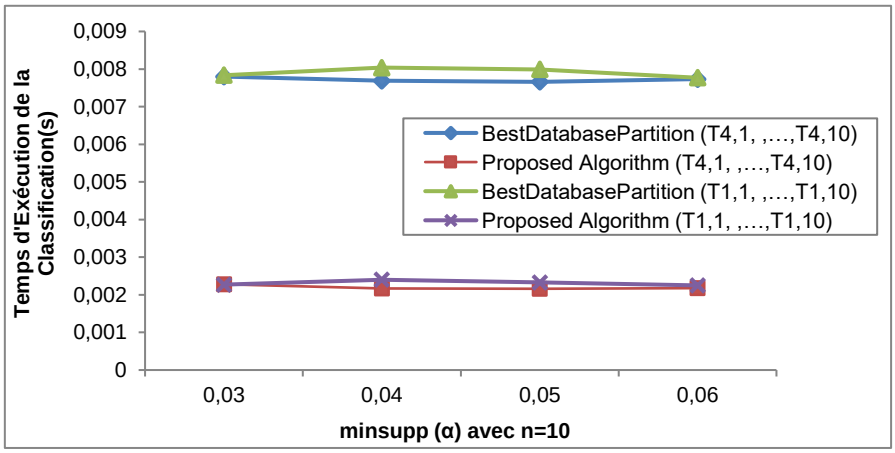



**FIGURE 4.2 -** TEMPS D'EXECUTION VS SUPPORT MINIMUM(ALPHA)

A partir des résultats obtenus dans l'étude I, nous pouvons constater que le temps d'exécution reste relativement constant pour les deux algorithmes quand la valeur de α augmente. La raison est que *n* et le nombre de classifications candidates (*m*=45) n'ont pas changé durant les expérimentations, comme cela est décrit dans la table 4.3. Cependant, le temps d'exécution de notre algorithme est plus court que celui de *BestDatabasePartition* [Adhikari et al. 2010a] pour α variant de 0.03 à 0.06.

De plus, nous observons que notre algorithme et *BestDatabasePartition* [Adhikari et al. 2010a] trouvent la même classification pour les *n* multiple bases de données. Cela est dû au fait que la probabilité qu'un itemset soit fréquent dans une base et qu'il soit non fréquent (e.i., présent avec un support non nul inférieur à minsupp(α)) dans une autre base est relativement petite dans les multi-bases testées.

## 3 Etude expérimentale II

Dans la deuxième étude expérimentale, nous avons analysé comment le nombre de bases de données *n* et le nombre de classifications candidates *m* influent sur la complexité temporelle des deux algorithmes. Ainsi, nous fixons la valeur de α=0.03 et nous faisons varier *n* de 3 à 20 bases de données. Les résultats d'expérimentations obtenus par les deux algorithmes sont présentés dans la table 4.4 et la figure 4.3.

| Ensembles de bases de données ($i$=1,4) | # de bases de données ($n$) | # de classifications candidates ($m$) à partir de | | Valeur *goodness* de la meilleure classification | | Temps d'exécution estimatif | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | Algorithme proposé $O(n^2)$ | *BestDatabasePartiton* $O(m \times n^2)$ | |
| | | $T_{1,3},\ldots,T_{1,n}$ | $T_{4,3},\ldots,T_{4,n}$ | $T_{1,3},\ldots,T_{1,n}$ | $T_{4,3},\ldots, T_{4,n}$ | $T_{1,3},\ldots, T_{1,n}$ $T_{4,3},\ldots, T_{4,n}$ | $T_{1,3},\ldots,T_{1,n}$ | $T_{4,3},\ldots,T_{4,n}$ |
| $T_{i,3},\ldots,T_{i,n}$ | 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 |

**Table 4.4 –** Comparaison des résultats de classification obtenus par notre algorithme et *BestDatabasePartition* (Adhikari et al., 2010a) sur les multiple bases de données $T_{1,1},\ldots, T_{1,10}$ et $T_{4,1},\ldots, T_{4,10}$ sous $\alpha$=0.03 pour $n$=3 to 20.

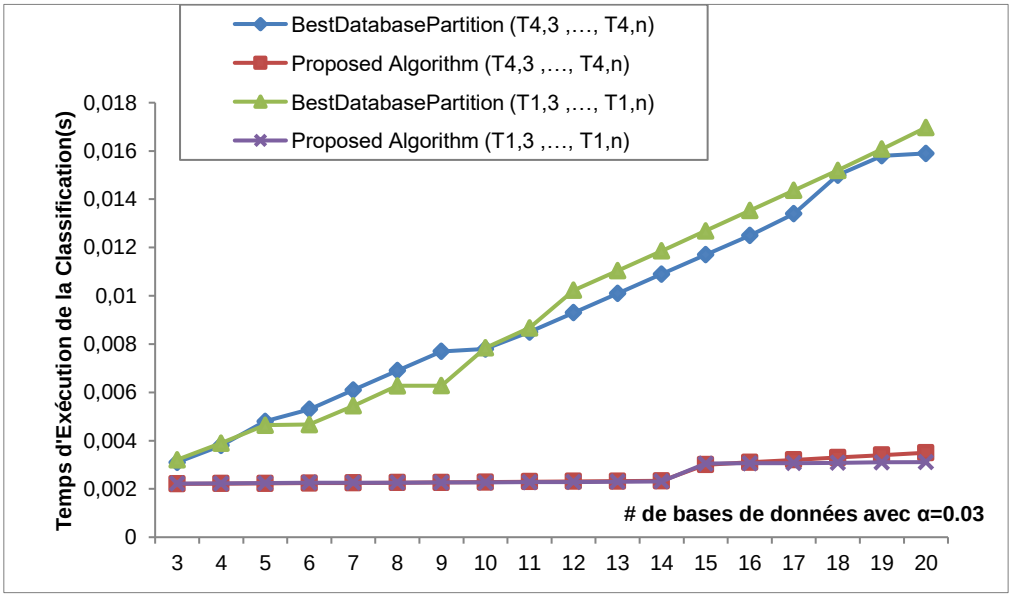



**FIGURE 4.3 -** LE TEMPS D'EXECUTION VS LE NOMBRE DE BASES DE DONNEES (N)

## 4 Analyse des résultats expérimentaux de l'étude II

A partir des résultats obtenus dans l'étude II, nous pouvons voir que le temps d'exécution tend à croitre rapidement lorsque la valeur de *n* croit elle aussi. Ce qui est le cas pour l'algorithme *BestDatabasePartition* [Adhikari et al. 2010a]. La raison est que *m* devient plus grand avec la croissance de *n*, comme le montre la table 4.4. Puisque la complexité temporelle de *BestDatabasePartition* dépend fortement de *m*, nous notons une croissance rapide de son temps d'exécution pour *n*= 3 to 20.

Contrairement aux algorithmes de classification existants [Adhikari et al. 2010a] [Li et al. 2009] [Liu et al. 2013] [Wu et al. 2005]**,** la complexité temporelle de notre algorithme dépend seulement du nombre de bases de données (*n*). En effet, en tout, ($n^2$-*n*)/2 arêtes (dans le graphe de base de données) sont explorées pour trouver la meilleure classification de l'ensemble des *n* multi-bases de données. Par conséquent, le temps d'exécution de notre algorithme est le plus court.

## 5 Etude expérimentale III

Pour consolider les résultats obtenus, les algorithmes existants [Wu et al. 2005], [Li et al. 2009], [Adhikari et al. 2010a] et [Miloudi et al. 2018] ont été implémentés en JAVA puis exécutés sur des tables de similarités issues de l'article [Li et al. 2009].

La troisième partie des expérimentations consiste à comparer les algorithmes en matière de la classification générée, temps CPU (CPU time) et utilisation de la mémoire (Memory Usage). De même, chaque algorithme est exécuté 1000 fois sur les mêmes tables de similarité pour avoir le temps CPU moyen en seconde et la mémoire utilisée moyenne en mégaoctets.

Les figures 4.4, 4.5 et 4.6 représentent les résultats expérimentaux de l'exécution des algorithmes [Wu et al. 2005] *en rouge***,** [Li et al. 2009] *en bleu***,** [Adhikari et al. 2010a] *en jaune* et [Miloudi et al. 2018] *en vert,* sur les tables de similarité table 4.5 , table 4.6 et table 4.7 respectivement.

Les méthodes JAVA suivantes ont été utilisées pour obtenir le temps d'exécution et la mémoire utilisée par les algorithmes :

*System.nanoTime*() : permet d'avoir le temps courant en nanoseconde ,

*Runtime.getRuntime*().*totalMemory* () *:* permet d'avoir la capacité (en octet) de la mémoire totale allouée au processus,

*Runtime.getRuntime().freeMemory*() : permet d'avoir la capacité (en octet) de la mémoire libre (non utilisée).

| ***sim*** | **DB1** | **DB2** | **DB3** | **DB4** | **DB5** | **DB6** | **DB7** | **DB8** |
|---|---|---|---|---|---|---|---|---|
| **DB1** | 1 | 0.4 | 0.56 | 0.4 | 0.5 | 0.4 | 0.1 | 0.2 |
| **DB2** | 0.4 | 1 | 0.4 | 0.79 | 0.4 | 0.45 | 0.2 | 0.1 |
| **DB3** | 0.56 | 0.4 | 1 | 0.4 | 0.45 | 0.4 | 0.15 | 0.1 |
| **DB4** | 0.4 | 0.79 | 0.4 | 1 | 0.4 | 0.45 | 0.2 | 0.1 |
| **DB5** | 0.5 | 0.4 | 0.45 | 0.4 | 1 | 0.4 | 0.1 | 0.1 |
| **DB6** | 0.4 | 0.45 | 0.4 | 0.45 | 0.4 | 1 | 0.2 | 0.2 |
| **DB7** | 0.1 | 0.2 | 0.15 | 0.2 | 0.1 | 0.2 | 1 | 0.28 |
| **DB8** | 0.2 | 0.1 | 0.1 | 0.1 | 0.1 | 0.2 | 0.28 | 1 |

**Table 4.5 –**La table de similarité entre 8 bases de données transactionnelles.

| ***sim*** | **DB1** | **DB2** | **DB3** | **DB4** | **DB5** | **DB6** |
|---|---|---|---|---|---|---|
| **DB1** | 1 | 0.952 | 0 | 0 | 0 | 0 |
| **DB2** | 0.952 | 1 | 0 | 0 | 0 | 0 |
| **DB3** | 0 | 0 | 1 | 0.813 | 0.584 | 0.486 |
| **DB4** | 0 | 0 | 0.813 | 1 | 0.727 | 0.544 |
| **DB5** | 0 | 0 | 0.584 | 0.727 | 1 | 0.652 |
| **DB6** | 0 | 0 | 0.486 | 0.544 | 0.652 | 1 |

**Table 4.6 –**La table de similarité entre 6 bases de données transactionnelles (a).

| *sim* | DB1 | DB2 | DB3 | DB4 | DB5 | DB6 |
|---|---|---|---|---|---|---|
| **DB1** | 1 | 0.667 | 0.429 | 0.333 | 0 | 0.111 |
| **DB2** | 0.667 | 1 | 0.667 | 0.333 | 0 | 0 |
| **DB3** | 0.429 | 0.667 | 1 | 0.538 | 0 | 0 |
| **DB4** | 0.333 | 0.333 | 0.538 | 1 | 0 | 0 |
| **DB5** | 0 | 0 | 0 | 0 | 1 | 0.333 |
| **DB6** | 0.111 | 0 | 0 | 0 | 0.333 | 1 |

**Table 4.7 –**La table de similarité entre 6 bases de données transactionnelles (b).

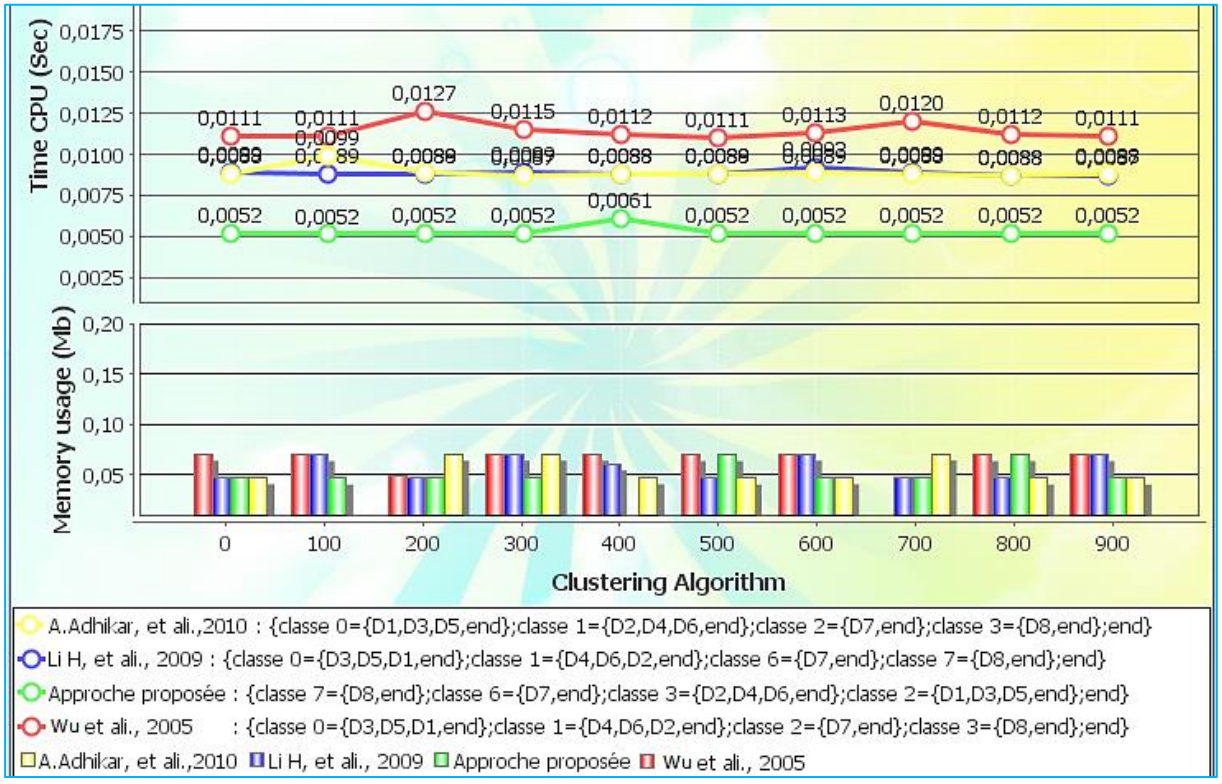



**FIGURE 4.4**– RESULTATS EXPERIMENTAUX DE L'EXECUTION DES ALGORITHMES SUR LA TABLE DE SIMILARITE **TABLE 4.5**

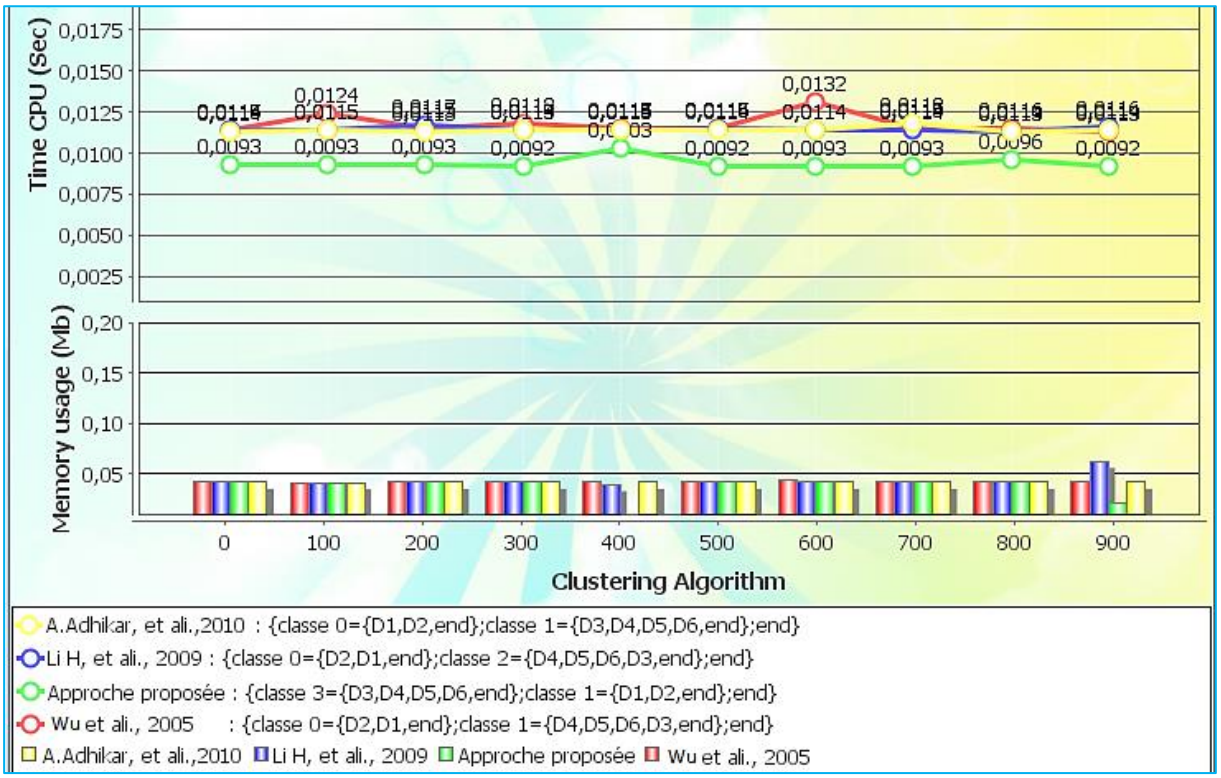



**FIGURE 4.5**– RESULTATS EXPERIMENTAUX DE L'EXECUTION DES ALGORITHMES SUR LA TABLE DE SIMILARITE **TABLE 4.6**

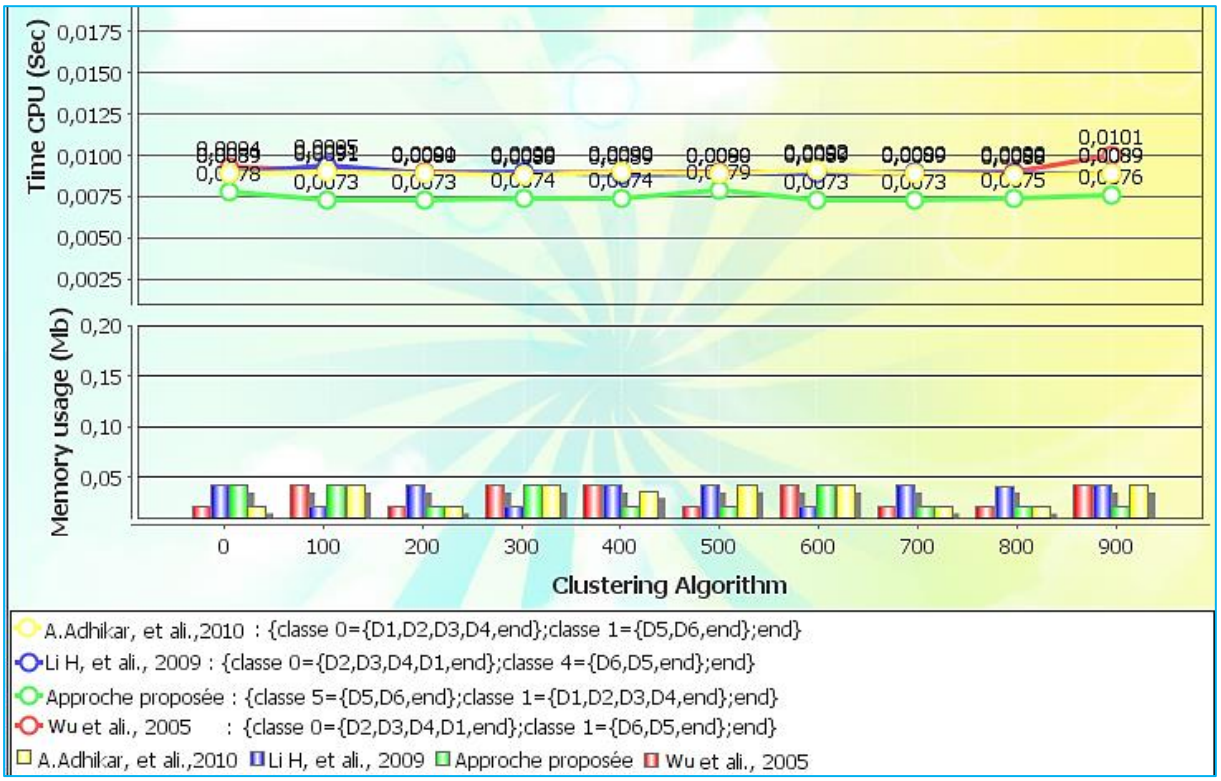



**FIGURE 4.6**– RESULTATS EXPERIMENTAUX DE L'EXECUTION DES ALGORITHMES SUR LA TABLE DE SIMILARITE **TABLE 4.7**

## 6 Analyse des résultats expérimentaux de l'étude III

Comme on peut le constater dans les légendes des figures 4.4, 4.5 et 4.6, les algorithmes étudiés génèrent la même classification pour la même table de similarité. Ceci est dû au fait que les algorithmes ont été exécutés sur les mêmes tables de similarités pour trouver les corrélations inter bases de données.

A partir de la table de similarité table 4.5**,** on peut voir qu'il existe $\boldsymbol{n}$=8 bases de données à classer et 9 valeurs de similarité distinctes {0.79, 0.56, 0.50, 0.45, 0.4, 0.28, 0.2, 0.15, 0.1}. Donc, en tout, $\boldsymbol{m}$=9 classifications candidates seront générées pour trouver la meilleure classification des $\boldsymbol{n}$ bases de données.

Pour chaque classification candidate (parmi les $\boldsymbol{m}$ classifications candidates), chaque algorithme parcourt les $\boldsymbol{n}$ bases de données un certain nombre de fois : $\boldsymbol{n^4}$ fois pour chaque classification générée par l'algorithme [Wu et al. 2005] *en rouge*, $\boldsymbol{n^3}$ fois pour chaque classification générée par l'algorithme [Li et al. 2009] *en bleu* et $\boldsymbol{n^2}$ fois pour chaque classification générée par l'algorithme [Adhikari et al. 2010a] *en jaune*. Cependant, pour notre algorithme [Miloudi et al. 2018] *en vert*, il suffit juste de parcourir les $\boldsymbol{n^2}$ arêtes du graphe de bases de données $G=(D,E)$ pour générer **toutes** les $\boldsymbol{m}$ classifications candidates et ainsi trouver la meilleure classification des $\boldsymbol{n}$ bases de données.

C'est pourquoi notre algorithme dans la figure 4.4 trouve la meilleure classification en un temps moyen optimal comme le montre la ligne du temps d'exécution *verte* qui reste toujours sous les autres lignes associées aux autres algorithmes [Wu et al. 2005], [Li et al. 2009] et [Adhikari et al. 2010a].

Idem pour la table 4.6, où il existe $\boldsymbol{n}$=6 bases de données à classer et $\boldsymbol{m}$=7 valeurs de similarité distinctes {0.952, 0.813, 0.727, 0.652, 0.584, 0.544, 0.486}. Et pour la table 4.7, où il existe $\boldsymbol{n}$=6 bases de données à classer et $\boldsymbol{m}$=5 valeurs de similarité distinctes {0.667, 0.538, 0.429, 0.333, 0.111}.

En analysant les résultats obtenus à partir des figures 4.5 et 4.6, on peut constater qu'à chaque fois le nombre de classifications candidates $\boldsymbol{m}$ diminue, les temps d'exécution des différents algorithmes deviennent de plus en plus courts et grandement dépendant de $\boldsymbol{n}$, ce qui explique pourquoi les lignes des temps d'exécution des différents algorithmes dans les figures 4.5 et 4.6 deviennent de plus en plus proches les unes des autres.

# Conclusion générale & Perspectives

## 1 Conclusion générale

Le nombre d'organisations multi-branches qui se trouvent dans le monde est en croissance considérable. Vu que la plupart de ces organisations possèdent de multiple bases de données distribuées à travers leurs différentes succursales, il devient important d'analyser ces multi-bases afin d'extraire des connaissances diverses et utiles pour le processus de prise de décision.

Cependant, l'analyse des données issues des multi-bases ne peut être réalisée avec succès sans un prétraitement efficace des données. En effet, une des techniques utilisée durant la préparation des données est le clustering ou la classification non supervisée. Le but est de trouver les groupes de bases de données les plus pertinents reflétant les mêmes caractéristiques et ayant une forte corrélation. Une fois le processus de clustering achevé, les différents groupes de bases de données peuvent être analysés individuellement pour la découverte de motifs utiles à la prise de décisions au niveau de chaque groupe de succursales.

Quelques approches de clustering des multi-bases de données ont été proposées dans la littérature [Adhikari et al. 2010] [Li et al. 2009] [Liu et al. 2013] [Wu et al. 2005]. Ces approches diffèrent principalement les unes des autres dans les points suivants : (a) la mesure de similarité utilisée pour estimer la pertinence entre les bases de données, (b) l'algorithme utilisé pour chercher la meilleure classification et (c) la mesure utilisée pour évaluer chaque classification.

Pour trouver la meilleure classification d'un ensemble de $n$ bases de données, les algorithmes existants génèrent au plus $(n^2-n)/2$ classifications hiérarchiques, une classification par niveau de similarité. Puis chaque classification est évaluée pour sélectionner la meilleure classification qui maximise le plus la similarité entre les bases de données situées dans un même cluster et minimise le plus la similarité entre les bases de données situées dans des clusters différents.

La limite majeure de ces algorithmes de clustering est le faite qu'ils génèrent chaque classification indépendamment, sans tenir compte de la réutilisation des clusters générés

dans les niveaux précédents. Par conséquent, un coût additionnel inutile est ajouté au coût total pour trouver la meilleure classification. De plus, la mesure de similarité utilisée pour estimer la pertinence entre les bases de données ne prend pas en considération les motifs locaux non fréquents dans les multi-bases Bien qu'ils peuvent améliorer grandement la qualité de la synthèse des motifs globaux au niveau de chaque cluster de base de données.

Toutes ces observations nous ont motivé à concevoir un algorithme amélioré qui recherche la meilleure classification des multi-bases de données en analysant un graphe de similarité. L'utilisation d'une telle structure de données permet le traçage des clusters générés dans les niveaux précédents et ainsi facilite la réutilisation de ces derniers pour générer les clusters des niveaux suivants. En outre, l'algorithme proposé fait participer les motifs non fréquents lors du calcul de la similarité entre les multi-bases de données permettant ainsi d'améliorer la qualité du clustering.

Les expérimentations réalisées sur des bases de données synthétiques ont prouvées l'efficacité de l'approche proposée et le temps d'exécution de notre algorithme est le plus court comparé à celui des algorithmes existants.

## 2 Perspectives

Les travaux futurs envisagés pour améliorer le travail proposé seront autour des points suivants :

### 2.1 Amélioration de la précision de la mesure de similarité :

- Les mesures de similarité proposées par [Adhikari et al. 2010a; Li et al. 2009; Liu et al. 2013; Wu et al. 2005; YUAN et al. 2012] sont basées sur la mesure de Jaccard [Tan et al. 2002] qui effectuent des intersections entes les itemsets fréquents des bases de données afin de calculer la similarité entre les bases de données. Ceci peut être coûteux en matière temps d'exécution lorsque le nombre d'itemsets fréquents est grand. Une méthode efficace pour estimer l'ensemble d'intersection entre deux sous-ensembles de données est d'utiliser le *minhashing* tel qu'il est décrit dans le livre [Leskovec et al. 2014] . Comme perspective, nous voulons évaluer ces différentes techniques afin de générer des classifications optimales tenant compte de la contrainte temps d'exécution et espace mémoire.

- L'estimation des motifs non fréquents dans les multi-bases de données permettrait d'améliorer la qualité du clustering généré. Pour cela nous voulons évaluer les modèles probabilistes proposés dans (Calders and Goethals, 2005; Jaroszewicz et al., 2002; Jelinek, 1997; Khiat et al., 2014a; Mannila and Toivonen, 1996; Pavlov et al., 2000, 2003; Yun, 2007) afin d'améliorer la qualité de la mesure de similarité proposée.

**2.2 Amélioration de la performance et flexibilité de l'algorithme :**

- Proposer une version parallèle et scallable de l'algorithme pour faire face aux bases de données volumineuses.
- La prise en charge de différents types de données tels que les bases de données relationnelles, texte et multimédia.
- Exploration de nouveaux types d'algorithmes tels que les travaux proposés par [Khiat et al. 2014b] et aussi des techniques issues de l'apprentissage machine, en particulier, les réseaux de neurones profonds à multicouches.

# Annexe

## 1 Structures de données alternatives :

Dans cette annexe, nous allons présenter et analyser quelques algorithmes et structures de données qu'on peut utiliser pour vérifier l'existence des doublons dans la table de similarité et trier les valeurs distinctes trouvées.

### a) Table de hachage :

On peut construire une table de hachage pour trouver les arêtes de $G=(D,E)$ ayant le même poids et les mettre dans une même liste chainée. Bien que les collisions des clés (poids des arêtes) puissent exister, en utilisant une fonction de hachage facile à calculer et qui distribue les clés uniformément sur les entrées de la table de hachage on pourrait vérifier l'existence des doublons en un temps constant à la moyenne des cas. La fonction de hachage utilisée dans notre code est celle définie dans la classe Java, ***java.util.Hashtable,*** qui utilise l'opérateur logique ***XOR*** sur les deux moitiés de la représentation longue entière binaire de la clé. Puis le résultat obtenu est modulo divisé sur la taille ***M*** de la table de hachage.

**a-1) Algorithme :**

Soit $D=\{D_1, D_2,\ldots, D_n\}$ l'ensemble des multi-bases de données et $E$ l'ensemble des arêtes dans $G$. Soit la table de hachage $H_G=\{\bigcup_{k=1}^{m} < \delta_k, E_{\delta k} > \}$ telle que la clé $\delta_k$ représente le poids de la $k$-ième arête et $E_{\delta k}$ représente la liste des arêtes dont le poids est égale à $\delta_k$. L'algorithme suivant permet de créer la table de hachage $H_G$ :

**Pour** chaque couple de bases de données $(D_i, D_j) \in D^2$ i≠j **faire**
  $\delta_k = sim(D_i, D_j)$ // Le poids de l'arête $[D_i, D_j]$
  **Si** $H_G(\delta_k)$ existe (≠null) **alors**
    Insérer l'arête $[D_i, D_j]$ en tête de la liste $E_{\delta k}$
  **Sinon**
    Créer la tête de liste $E_{\delta k}$. headEdgeList = [Di, Dj]
    Ajouter l'entrée $< \delta_k, E_{\delta k} >$ dans $H_G$
  **Fin Si**
**Fin boucle Pour**

**FIGURE 6.1 -** ALGORITHME DE CREATION DE LA TABLE DE HACHAGE DES ARETES

Puisque il existe $(n^2-n)/2$ clés différentes au maximum, la taille de la table de hachage est initialiser à $(n^2-n)/2$. Avec une fonction de hachage facile à calculer et uniforme, les clés seront distribuées de façon équitable sur les entrées de la table et l'existence d'une clé peut être vérifiée en un temps constant. Chaque clé nécessite au plus $s^2$ comparaisons entre itemsets fréquents tel que $s$ est le nombre maximum d'itemsets fréquents dans une base de données transactionnelle. Ainsi, la *complexité temporelle* de la construction de $H_G$ est $O(n^2s^2)$. Le nombre d'unités requises pour stocker les itemsets fréquents est inférieur ou égale à $n*s$. Pour la construction de $H_G$, nous avons besoin de $m \in [1,(n^2-n)/2]$ unités pour sauvegarder les clés distinctes et $n^2$ unités pour sauvegarder les arêtes. Par conséquent, *la complexité spatiale* nécessaire pour la construction de $H_G$ est $O(n^2$*(unité arêtes)*$+n*s$*(unité itemset fréquent)*$+m$ *(unité similarité*)). Une fois la table $H_G$ est construite, ses $m$ entrées non vides $<\delta_k, E_{\delta k}>$ sont triées dans l'ordre décroissant des clés $\delta_k$ en utilisant un tri de complexité $O(m \times log_2(m))$. Les étapes de construction de la table de hachage $H_G$ et du tri des arêtes à partir de la table de similarité 3.1 sont illustrées dans la figure suivante :

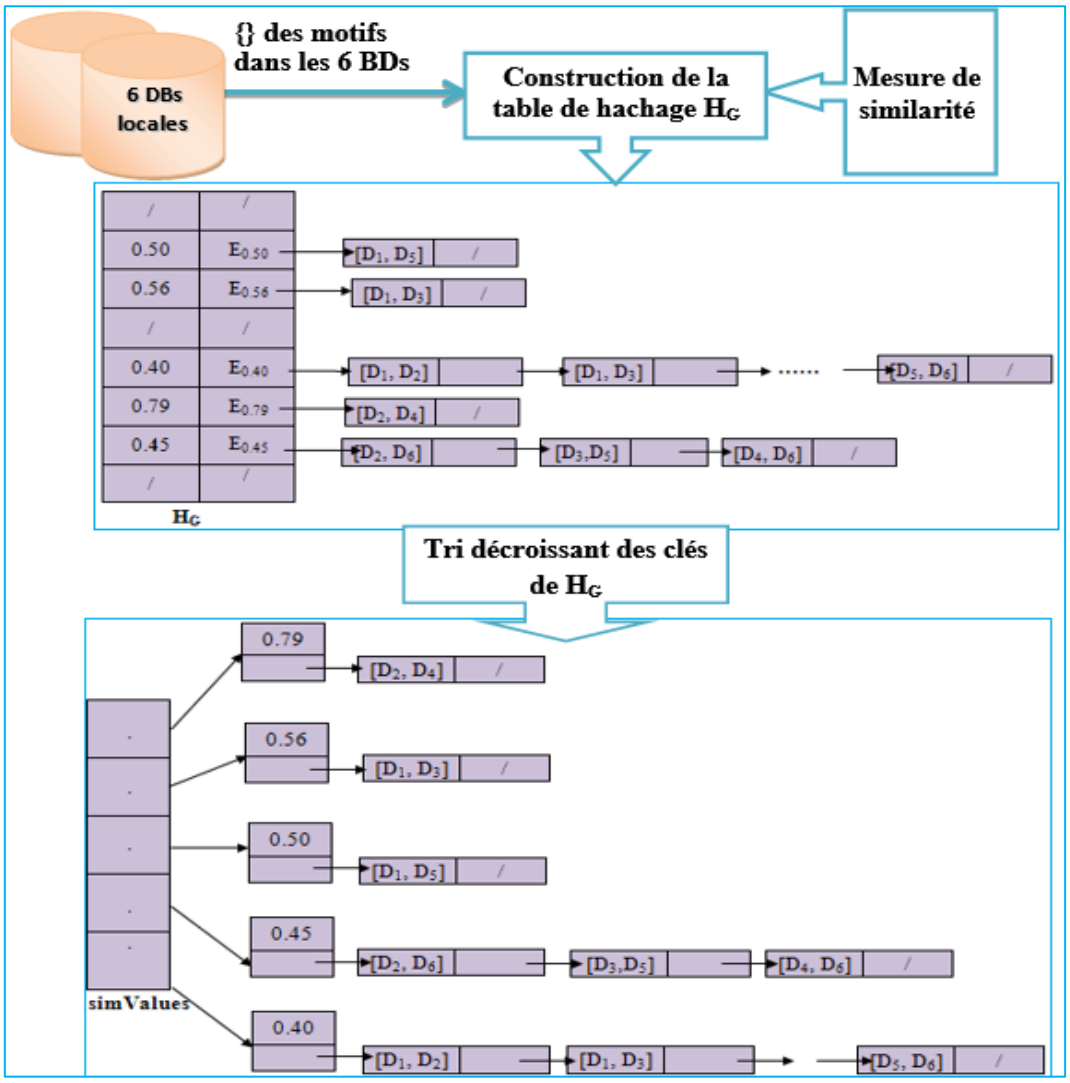



**FIGURE 6.2 -** CONSTRUCTION DE LA TABLE DE HACHAGE HG POUR L'EXEMPLE DANS **TABLE 3.1**

**b) Tri par fusion** [Cormen 2009] **:**

En utilisant le tri-par fusion, on peut trier les $m$ valeurs de similarité distinctes en un temps estimé à $O(m \times \log_2(m))$.

**c) Tri par base** [Cormen 2009] :

Il est considéré comme solution idéale pour trier les $(n^2-n)/2$ valeurs de la table de similarité situées dans l'intervalle [0,1]. La complexité temporelle de cet algorithme au pire des cas est $O(d(n^2+k))$ où $d$ est le nombre maximal de décimales après la virgule et chaque chiffre appartient à l'ensemble $\{0, 1,..., k\text{-}1\}$. Puisque $d$ et $k$ sont des constantes, le tri par base s'exécute en un temps linéaire par rapport à la taille de l'entrée.

Soit A[0..($n^2$-$n$)/2-1] l'ensemble des ($n^2$-$n$)/2 arêtes du graphe $G$=($D$,$E$). Alors, l'algorithme suivant permet de trier toutes les arêtes de $G$ dans l'ordre décroissant de leurs poids A[$j$].weight :

**Algorithme *RadixEdgeSort*(**$n$,$d$,$k$,A,B**)**

*Entrée* : $n$ : le nombre de bases de données;
$d$ : nombre de décimales après la virgule ;
$k$: plage des valeurs possibles des chiffres de 0 à $k$-1;
A[0..($n^2$-$n$)/2-1] :la table des arêtes ;
B[0..$k$-1] : table temporaire de paquets (listes chainées) des arêtes ;

*Sortie* : A[0..($n^2$-$n$)/2-1] : la liste des arêtes triées dans l'ordre décroissant de leurs poids ;

1. **Pour** $i$=1 **à** $d$ **faire** //Aller du chiffre le moins significatif au chiffre le plus significatif
2. **Pour chaque** élément A[$j$]≠nil dans A **faire**
3. $c$=((A[$j$].weight×$10^i$) mod $10^i$)/ $10^{i-1}$ //Obtenir le chiffre de rang $i$ dans A[$j$].weight;
4. Insérer A[$j$] en tête de la liste B[$c$] ;
5. A[$j$]=nil ;
6. **Fin boucle Pour**
7. **Pour** $j$=0 **à** $k$-1 **faire**
   //Mettre dans l'ordre les listes B[0], B[1],..., B[$k$−1] dans A[$k$-1], A[$k$-2],..., A[0] ;
8. A[$j$]= B[$k$-1-$j$] ;
9. B[$k$-1-$j$]= nil ;
10. **Fin boucle Pour**
11. **Fin boucle Pour**
12. **Retourner** A[1..($n^2$-$n$)/2] ;

**FIGURE 6.3 -** ALGORITHME ALTERNATIF 1 POUR LE TRI DES ARETES DU GRAPHE G.

Cet algorithme est efficace en matière complexité temporelle, puisque il permet de trier toutes les arêtes de $G$ en un temps linéaire, néanmoins il possède certaines limitations. En effet, le faite de choisir un nombre de décimales $d$ non adéquat ou petit peut éliminer certaines classifications et éviter dans ce cas de trouver la meilleure classification.

Une autre variante plus optimale de cet algorithme peut être utilisée lorsque $10^d$ < ($n^2$-$n$)/2. Dans ce cas on pourrait initialiser B à $10^d$ entrées et chaque arête de même poids sera insérée directement en tête de la liste B[$\lfloor A[j].weight \times 10^d \rfloor$]. L'algorithme est illustré comme suit :

**Algorithme *EdgeSortVariante*(*n*,*d*,A,B)**

*Entrée* : *n* : le nombre de bases de données;
*d* : nombre de décimales après la virgule ;
A[0..($n^2$-*n*)/2-1] :la table des arêtes ;
B[0.. $10^d$-1] : table temporaire de paquets (listes chainées) des arêtes ;

*Sortie* : A[1..($n^2$-*n*)/2] : la liste des arêtes triées dans l'ordre décroissant de leurs poids ;

1. **Pour chaque** élément A[*j*] dans A **faire**
2. *c*=A[*j*].weight×$10^d$;
3. Insérer A[*j*] en tête de la liste B[⌊*c*⌋] ;
4. //les arêtes de même poids seront insérées dans la même liste ;
5. **Fin boucle Pour**
6. *k*=0 ;
7. **Pour** *j*=$10^d$-1 **à** 0 **faire**
8. A[*k*]= B[*j*] ;
9. *k*=*k*+1;
10. **Fin boucle Pour**
11. **Retourner** A[1..($n^2$-*n*)/2] ;

**FIGURE 6.4 -** ALGORITHME ALTERNATIF 2 POUR LE TRI DES ARETES DU GRAPHE G

La complexité temporelle de l'algorithme est $O(10^d + n^2)$. Si $10^d < n^2$, alors cet algorithme est capable de trier les ($n^2$-*n*)/2 arêtes en un temps linéaire. L'avantage de cet algorithme est qu'il est indépendant du nombre d'arêtes puisque il dépend uniquement du nombre de décimales « ***d*** » après la virgule. Cependant, la meilleure valeur pour **« *d* »** ne peut pas être choisie à priori et donc $10^d$ peut être très grand que $n^2$, ce qui va engendrer une croissance du temps d'exécution en plus de l'espace mémoire requis pour trouver la meilleure classification.

En supposant que le nombre maximal de décimales après la virgule est *d=2*, la figure suivante illustre le déroulement de l'algorithme « alternatif 2 » sur la table de similarité 3.1.

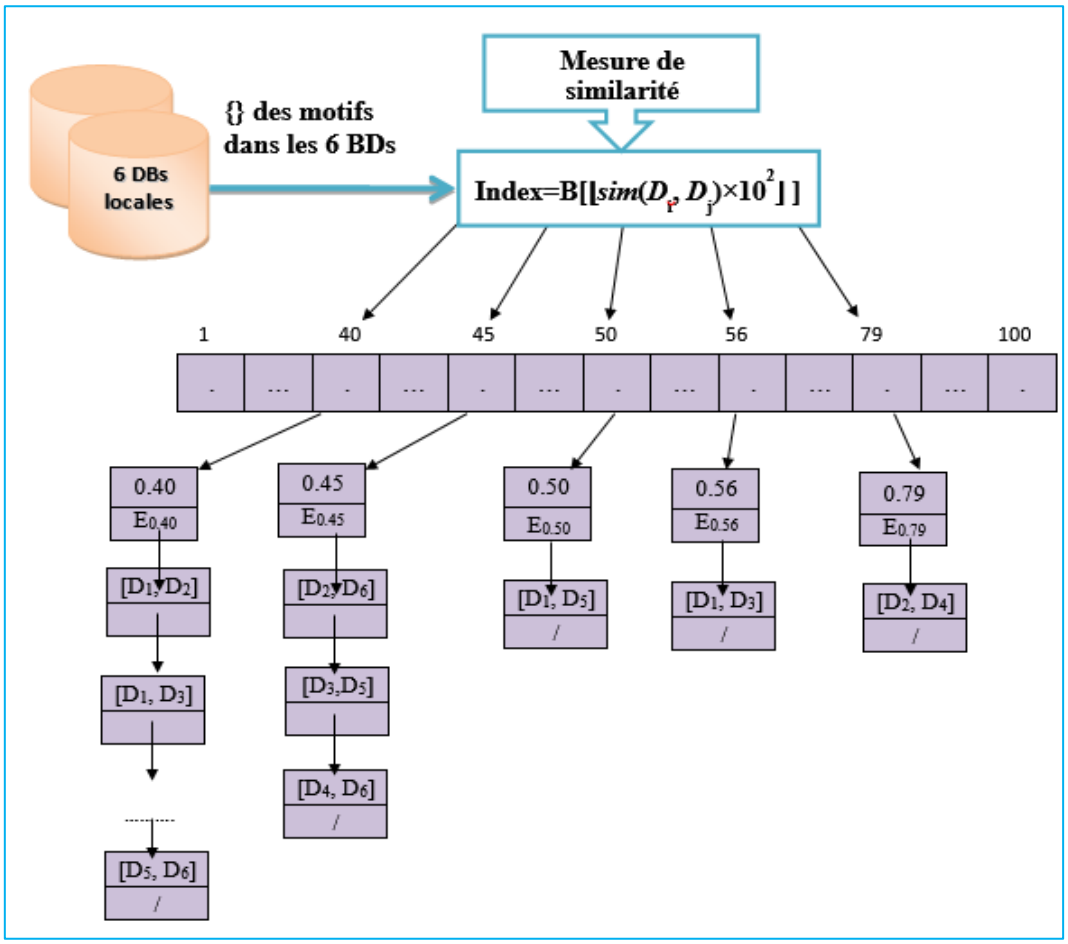



**Figure 6.5 –** Illustration de l'exécution de l'Algorithme alternatif 2 sur l'exemple dans **Table 3.1**

# Bibliographie

[ADHIKARI ET AL. 2010a] Adhikari, A., Ramachandrarao, P., and Pedrycz, W. Efficient Clustering of Databases Induced by Local Patterns. In: *Developing Multi-Database Mining Applications*. Springer, London, 95–120, 2010.

[ADHIKARI ET AL. 2010b] Adhikari, A., Ramachandrarao, P., and Pedrycz, W. *Developing multi-database mining applications*. Springer Science & Business Media, 2010.

[ADHIKARI AND RAO. 2008] Adhikari, A. and Rao, P.R. Synthesizing heavy association rules from different real data sources. *Pattern Recognition Letters 29*, 1, 59–71, 2008.

[AGRAWAL AND SHAFER. 1996] Agrawal, R. and Shafer, J.C. Parallel mining of association rules. *IEEE Transactions on knowledge and Data Engineering 8*, 6, 962–969, 1996.

[AGRAWAL ET AL. 1994] Agrawal, R., Srikant, R., and others Fast algorithms for mining association rules. *Proc. 20th int. conf. very large data bases, VLDB*, 487–499, 1994.

[CALDERS AND GOETHALS. 2005] Calders, T. and Goethals, B. Quick inclusion-exclusion. *International Workshop on Knowledge Discovery in Inductive Databases*, Springer, 86–103, 2005.

[CORMEN. 2009] Cormen, T.H. *Introduction to algorithms*. MIT press, 2009.

[HAN ET AL. 2000] Han, J., Pei, J., and Yin, Y. Mining frequent patterns without candidate generation. *ACM sigmod record*, ACM, 1–12, 2000.

[JAROSZEWICZ ET AL. 2002] Jaroszewicz, S., Simivici, D.A., and Rosenberg, I. An inclusion-exclusion result for boolean polynomials and its applications in data mining. *Proc. of the Discrete Mathematics in Data Mining Workshop, SIAM Datamining Conference*, Citeseer, 2002.

[JELINEK. 1997] Jelinek, F. *Statistical methods for speech recognition*. MIT press, 1997.

[KHIAT. 2015] Khiat, S. LA FOUILLE MULTI-SOURCES DE DONNEES MULTI-NIVEAUX. https://www.univ-usto.dz/images/coursenligne/these_khiat_salim.pdf. *Thèse de Doctorat 2015.*

[KHIAT ET AL. 2014a] Khiat, S., Belbachir, H., and Rahal, S.A. Probabilistic Models for Local Patterns Analysis. *Journal of Information Processing Systems 10*, 1, 145–161, 2014.

[KHIAT ET AL. 2014b] Khiat, S., Belbachir, H., and Rahal, S.A. Multi-Level Synthesis of Frequent Rules from Different Databases Using A Clustering Approach. *Proceedings of the International Conference on Data Mining (DMIN)*, 2014.

[LESKOVEC ET AL. 2014] Leskovec, J., Rajaraman, A., and Ullman, J.D. *Mining of massive datasets*. Cambridge university press, 2014.

[LI ET AL. 2009] Li, H., Hu, X., and Zhang, Y. An Improved Database Classification Algorithm for Multi-database Mining. *Frontiers in Algorithmics*, Springer, Berlin, Heidelberg, 346–357, 2009.

[LIU ET AL. 1998] Liu, H., Lu, H., and Yao, J. Identifying relevant databases for multidatabase mining. *Research and Development in Knowledge Discovery and Data Mining*, Springer, Berlin, Heidelberg, 210–221, 1998.

[LIU ET AL. 2001] Liu, H., Lu, H., and Yao, J. Toward multidatabase mining: identifying relevant databases. *IEEE Transactions on Knowledge and Data Engineering 13*, 4, 541–553, 2001.

[LIU ET AL. 2013] Liu, Y., Yuan, D., and Cuan, Y. Completely Clustering for Multi-databases Mining. *Journal of Computational Information Systems 9, 16*, 2013.

[MANNILA AND TOIVONEN. 1996] Mannila, H. and Toivonen, H. Multiple uses of frequent sets and condensed representations: Extended abstract. *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, 189–194, 1996.

[MILOUDI ET AL. 2016a] Miloudi, S., Hebri, S.A.R., and Khiat, S. An improved multi-database classification approach. *Proceedings of Engineering & Technology (PET)*, 610–616, 2016.

[MILOUDI ET AL. 2016b] Miloudi, S., Hebri, S.A.R., and Khiat, S. Multi-database Classification Approaches : A Literature Review. *Actes du Colloque COSI'2016*, 158–169, 2016.

[MILOUDI ET AL. 2018] Miloudi, S., Hebri, S.A.R., and Khiat, S. Contribution to Improve Database Classification Algorithms for Multi-Database Mining. *JOURNAL OF INFORMATION PROCESSING SYSTEMS 14*, 3, 709–726, 2018.

[MOTWANI AND RAGHAVAN. 2010] Motwani, R. and Raghavan, P. *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[NA ET AL. 2010] Na, S., Xumin, L., and Yong, G. Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm. *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, 63–67, 2010.

[PARK ET AL. 1995] Park, J.S., Chen, M.-S., and Yu, P.S. Efficient parallel data mining for association rules. *Proceedings of the fourth international conference on Information and knowledge management*, ACM, 31–36, 1995.

[PARTHASARATHY ET AL. 2001] Parthasarathy, S., Zaki, M.J., Ogihara, M., and Li, W. Parallel data mining for association rules on shared-memory systems. *Knowledge and Information Systems 3*, 1, 1–29, 2001.

[PAVLOV ET AL. 2000] Pavlov, D., Mannila, H., and Smyth, P. Probabilistic models for query approximation with large sparse binary data sets. *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 465–472, 2000.

[PAVLOV ET AL. 2003] Pavlov, D.N., Mannila, H., and Smyth, P. Beyond independence: Probabilistic models for query approximation on binary transaction data. *IEEE Transactions on Knowledge and Data Engineering 15*, 6, 1409–1421, 2003.

[PRODROMIDIS AND STOLFO. 1998] Prodromidis, A. and Stolfo, S.J. Pruning meta-classifiers in a distributed data mining system. *Proceedings of the First National Conference on New Information Technologies*, 1998.

[RAMKUMAR AND SRINIVASAN. 2008] Ramkumar, T. and Srinivasan, R. Modified algorithms for synthesizing high-frequency rules from different data sources. *Knowledge and information systems 17*, 3, 313–334, 2008.

[RAMKUMAR AND SRINIVASAN. 2010] Ramkumar, T. and Srinivasan, R. Multi-Level Synthesis of Frequent Rules from Different Data-Sources. *International Journal of Computer Theory and Engineering 2*, 2, 195, 2010.

[SRINIVASAN AND RAMKUMAR. 2009] Srinivasan, R. and Ramkumar, T. The effect of correction factor in synthesizing global rules in a multi-database mining scenario. *Journal of Applied Computer Science & Mathematics 3*, 6, 33–38, 2009.

[TAN ET AL. 2002] Tan, P.-N., Kumar, V., and Srivastava, J. Selecting the right interestingness measure for association patterns. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 32–41, 2002.

[TOIVONEN AND OTHERS. 1996] Toivonen, H. and others Sampling large databases for association rules. *VLDB*, 134–145, 1996.

[WU ET AL. 2005] Wu, X., Zhang, C., and Zhang, S. Database classification for multi-database mining. *Information Systems 30*, 1, 71–88, 2005.

[WU AND ZHANG. 2003] Wu, X. and Zhang, S. Synthesizing high-frequency rules from different data sources. *IEEE Transactions on Knowledge and Data Engineering 15*, 2, 353–367, 2003.

[YIN AND HAN. 2005] Yin, X. and Han, J. Efficient classification from multiple heterogeneous databases. *European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, 404–416, 2005.

[YUAN ET AL. 2012] YUAN, D., FU, H., LI, Z., and WU, H. An application-independent database classification method based on high cohesion and low coupling. *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE 9*, 15, 4337–4344, 2012.

[YUN. 2007] Yun, U. Efficient mining of weighted interesting patterns with a strong weight and/or support affinity. *Information Sciences 177*, 17, 3477–3499, 2007.

[ZHANG ET AL. 2003] Zhang, S., Wu, X., and Zhang, C. Multi-database mining. *IEEE Computational Intelligence Bulletin 2*, 1, 5–13, 2003.

[ZHANG ET AL. 2009] Zhang, S., You, X., Jin, Z., and Wu, X. Mining globally interesting patterns from multiple databases using kernel estimation. *Expert Systems with Applications 36*, 8, 10863–10869, 2009.

[ZHANG AND ZAKI. 2006] Zhang, S. and Zaki, M.J. Mining Multiple Data Sources: Local Pattern Analysis. *Data Mining and Knowledge Discovery 12*, 2, 121–125, 2006.

[ZHANG ET AL. 1997] Zhang, T., Ramakrishnan, R., and Livny, M. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery 1*, 2, 141–182, 1997.

# LISTE DES PUBLICATIONS

**« Contribution to Improve Database Classification Algorithms for Multi-Database Mining »**. Journal of Information and Processing Systems (JIPS). e-ISSN : **2092-805X(Electronic) ;** p-ISSN : **1976-913X(Print). Vol.14, No.3, pp.709-726, June 2018.**

Lien : **http://jips-k.org/q.jips?cp=pp&pn=570**

# COMMUNICATIONS

**« An improved multi-database classification approach »** in Proceedings of Engineering and Technology (PET). **Vol.13, Special issue, pp.610-616, March 2016.**

Lien 1 : **https://www.researchgate.net/publication/315670332_An_improved_multi-database_classification_approach**

Lien 2 : **http://ipco-co.com/PET_Journal/Volume22_ACECS_2016.html**

**«Multi-database Classification Approaches : A Literature Review »** au Colloque sur l'Optimisation et les Systèmes d'Information (COSI) à Sétif. **Mai 2016.**

Lien : **https://www.researchgate.net/publication/326200756_Multi-database_Classification_Approaches_A_Literature_Review**