Liste des abréviations

ECD	**E**xtraction de **C**onnaissances à partir de **D**onnées
KDD	**K**nowledge **D**iscovery in **D**atabases
DM	**D**ata **M**ining
FDD	**F**ouille **D**e **D**onnées
EPDC	**E**xtraction **P**arallèle et **D**istribuée de **C**onnaissances
PDKD	**P**arallel and **D**istributed **K**nowledge **D**iscovery
DMPD	**D**ata **M**ining **P**arallèle et **D**istribué
PDDM	**P**arallel and **D**istributed **D**ata **M**ining
ED	**E**ntrepôt de **D**onnées
DW	**D**ata **W**arehouse
DMs	**D**ata **M**arts
ODS	**O**perational **D**ata **S**tore
OLAP	**O**n-**L**ine **A**nalytical **P**rocessing
OLTP	**O**n-**L**ine **T**ransaction **P**rocessing
ROLAP	**R**elational **OLAP**
MOLAP	**M**ultidimensional **OLAP**
HOLAP	**H**ybrid **OLAP**
TID	**T**ransaction **ID**entification
ETL	**E**xtract-**T**ransform-**L**oad
k-PPV	*k*-**P**lus **P**roches **V**oisins
k-NN	***k***-**N**earest **N**eighbor
CHA	**C**lassification **H**iérarchique **A**scendante
RNA	**R**éseau de **N**eurones **A**rtificiel
ANN	**A**rtificial **N**eural **N**etwork
PMC	**P**erceptron **M**ulti**C**ouche

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction Générale

Contexte et problématiques

Dans le contexte économique concurrentiel de nos jours, l'information joue un rôle crucial dans le quotidien des entreprises. L'acquisition, l'analyse et l'exploitation des informations sont devenues des choix stratégiques incontournables. La maîtrise de l'information est une compétence capitale pour toute entreprise voulant s'imposer dans les premiers rangs de son domaine d'activité. À la lumière de ces impératifs, les grands volumes de données de production, relatifs à l'activité de l'entreprise, sont devenus de véritables mines de connaissances. À partir de ce moment, de gros efforts sont à déployer pour maîtriser les grandes masses de données d'une part et pour extraire des connaissances potentielles à partir de ces données d'autre part.

Les entrepôts de données (*Data Warehouses*) ont apportés une solution adéquate et efficace au problème du stockage et de la gestion de données. Un entrepôt est une base centralisée de grands volumes de données, historisées, organisées par sujet et consolidées à partir de divers systèmes de production de l'entreprise. En plus de sa vocation de stockage, la modélisation d'un entrepôt est complètement dédiée à l'analyse de ces données. En effet, l'entrepôt de données constitue un support pour les outils d'analyse en ligne issue de la technologie OLAP (*On-Line Analytical Processing*). Ces outils offre des possibilités pour visualiser, structurer et explorer les données de l'entrepôt, mais ne permet pas d'extraire les connaissances potentielles contenues dans ces données. D'une manière générale, dans ce processus d'aide à la décision, c'est à l'utilisateur de trouver manuellement, en utilisant les outils de l'analyse en ligne, les connaissances cachées dans les données de l'entrepôt.

D'autre côté, l'Extraction de Connaissances à partir des bases de Données (ECD), ou « *Knowledge Discovery in Databases* (*KDD*) » en anglais, est un nouveau domaine de recherche qui tente de répondre aux besoins d'analyser et d'extraire les connaissances cachées dans les grands volumes de données. L'ECD a été définie comme le « processus non trivial d'identification dans les grandes masses de données des structures inconnues, valides et potentiellement exploitables ». Grâce aux bons résultats obtenus et à la maturité des techniques pour l'extraction de connaissances, l'ECD a dépassé son domaine d'origine (les entreprises) pour être employé dans divers champs d'application tels que la médecine la

biologie, la finance, etc. L'ECD est un processus interactif et itératif composé de différentes étapes groupées globalement dans les quatre étapes suivantes : la collection de données (*Data Warehousing*), la préparation de donnée (*Pre-processing*), le fouille de données (*Data Mining*) et la validation des modèles ainsi élaborés (*Post-traitement*).

L'étape de Data Mining est véritablement celle où l'on cherche à inférer de nouvelles connaissances en appliquant des méthodes dites de Data Mining. Le Data Mining fait appel à un lot de méthodes issues de la statistique, de l'analyse de données, de la reconnaissance des formes, de l'intelligence artificielle ou de l'apprentissage automatique afin d'induire des modèles de connaissances exprimés dans des formalismes valides et compréhensibles. Dans la littérature, on distingue généralement trois grandes familles de techniques de Data Mining : les techniques de recherche d'association, les techniques de classification et les techniques de segmentation ou de Clustering.

D'une manière parallèle à l'évolution des entrepôts de données et des techniques d'extraction de connaissances (ou de Data Mining), nous assistons ces dernières années à une décentralisation de système décisionnel de l'entreprise. En effet, les données de l'entrepôt de l'entreprise sont souvent réparties dans des magasins de données (*Data Marts*), dédiées à un domaine d'activité ou à un département particulier de l'entreprise, qui sont généralement dispersés sur des sites éloignés géographiquement. Face au problème de distribution de données, les chercheurs de domaine de l'ECD sont de plus en plus conscients de l'utilité de développement des solutions parallèles et distribuées d'extraction de connaissances, d'où une nouvelle branche dans le domaine de l'ECD a été révélée, l'Extraction Parallèle et Distribuée de Connaissances (EPDC), ou encore « *Parallel and Distributed Knowledge Discovery (PDKD)* » en anglais.

Le contexte de notre problématique s'inscrit justement de cette optique. Il s'agit plus précisément d'étudier le concept d'entrepôt de données et les différentes techniques d'extraction de connaissances à partir de données, et de proposer par la suite une solution parallèle et distribuée quant à l'extraction de connaissances à partir des entrepôts de données répartis. Dans notre travail, on s'intéresse plus précisément aux connaissances extraites sous forme de règles d'association, en raison de leurs simplicités (facilement interprétable en connaissances utiles) et la simplicité de ces algorithmes.

Contributions

Dans ce travail, nous présentons une étude détaillée sur la problématique d'extraction parallèle et distribuée de connaissances à partir des entrepôts de données, puis nous proposons une approche parallèle et distribuée d'extraction de connaissances exprimables sous forme de règles d'association à partir d'un entrepôt de données réparti.

Dans un premier temps, nous proposons un algorithme de génération des itemsets fréquents. Cet algorithme présente une amélioration pour l'algorithme Partition ; l'idée de base est d'exécuter l'algorithme Partition d'une façon incrémentale et à chaque traitement d'une nouvelle partition, l'algorithme mis à jour ces connaissances (itemsets fréquents, itemsets non fréquents et itemsets candidats) sur les données traitées. En effet, cette démarche incrémentale nous a permet d'exploiter au maximum les calculs effectués durant la première phase du l'algorithme, en générant le maximum d'itemsets fréquents et non fréquents toute en réduisant le nombre d'itemsets candidats pour la deuxième phase du l'algorithme.

Nous proposons, également, un algorithme de mise à jour des itemsets fréquents à exploiter en cas de rafraîchissement de l'entrepôt de données. Cet algorithme présente une amélioration pour l'algorithme Incremental Mining, qui tente de pousser encore mieux l'étape de parcourt complet de l'entrepôt de données.

Par la suite, en se basant sur ces deux algorithmes, nous proposons une approche parallèle et distribuée d'extraction de règles d'association à partir d'un entrepôt de données réparti. Cette approche est constituée d'un ensemble d'agents mobiles et stationnaires coopérant pour calculer les itemsets localement fréquents sur chaque site du réseau, l'ensemble des itemsets globalement fréquents sur la totalité des données de l'entrepôt et enfin, l'ensemble de règles d'association sur l'entrepôt de données global. Également, l'approche proposée prend en considération le cas de mise à jour de l'entrepôt de données, en proposant une démarche constituée d'un ensemble d'agent mobiles et stationnaires coopérant pour mettre à jour ces connaissances au niveau local (itemsets localement fréquents) et au niveau global (itemsets globalement fréquents et règles d'association).

Organisation du mémoire

Ce mémoire s'articule autour de six chapitres organisés comme suit :

- **Le chapitre 1** décrit la terminologie liée au domaine de l'Extraction de Connaissances à partir de Données (ECD). Nous décrivons, également, le cycle complet de Découverte de Connaissances dans des bases de données en général, les différentes tâches de Data Mining et ainsi que les différents supports de stockage pour le Data Mining.
- **Le chapitre 2** présente un état de l'art sur les entrepôts de données. Nous détaillons les différentes notions nécessaires à leurs compréhensions comme l'architecture, les stratégies de conception et de construction des entrepôts de données, la modélisation multidimensionnelle adoptée aux données de l'entrepôt, les opérations de manipulation des données de l'entrepôt et enfin, son processus d'alimentation.

- **Le chapitre 3** est consacré à l'étude de quelques techniques d'extraction de connaissances (ou de Data Mining). Nous décrivons les différentes techniques utilisées pour résoudre quelques problèmes de Data Mining à savoir, la classification, le Clustering et la recherche de règles d'association.
- **Le chapitre 4** adresse les problèmes d'extraction parallèle et distribuée des connaissances à partir des grands volumes de données distribués et hétérogènes. Les différents composants algorithmes de la conception parallèle et distribuée des techniques d'extraction de connaissances sont également présentés dans ce chapitre.
- **Le chapitre 5**, quant à lui est consacré, aux techniques parallèles et distribuées d'extraction de connaissances. Son objectif est d'étudier les différentes approches et stratégies de parallélisation et de distribution des techniques d'extraction de connaissances proposées dans la littérature.
- **Le chapitre 6** fait l'objet de notre nouvelle approche parallèle et distribuée d'extraction de connaissances. Dans un premier temps, nous décrivons l'architecture du réseau et la stratégie de distribution de données assumée par l'approche proposée pour l'extraction de règles d'association à partir d'un entrepôt de données réparti. Ensuite, nous détaillons le principe des algorithmes proposés pour la génération des itemsets fréquents et leurs mises à jour sur chaque site du réseau, l'algorithme « Calcul_Itemsets_Fréquents » et l'algorithme « Mise_à_jour_Itemsets_Fréquents » respectivement. Enfin, nous décrivons en détail le principe de l'approche proposée.
- **Dans la conclusion générale,** Nous rappelons les grands points qui ont été abordé et ainsi que des perspectives que nous souhaitons accomplir prochainement.
- **En annexe**, nous présentons une description détaillée de quelques notions liées aux systèmes multi-agents, aux agents mobiles et aux aglets (la plateforme la plus populaire des agents mobiles).

CHAPITRE 1

Introduction à l'Extraction de Connaissances à partir de Données

1. 1 Introduction

L'Extraction de Connaissances à partir de Données (ECD), ou « *Knowledge Discovery in Databases (KDD)* », est une jeune discipline apparue au début des années 90. Mais, les techniques utilisées dans le domaine de l'ECD remontent aux années 50, quand les travaux des mathématiciens, des logiciens et des informaticiens sont combinés pour créer de l'Intelligence Artificielle (IA) et de l'Apprentissage Automatique [Buc 2006]. L'émergence de l'ECD est principalement due au développement des moyens informatiques de stockage tels que les entrepôts de données (*Data Warehouses*) et de calcul tels que les grilles de calcul (*Grid Computing*).

Le Data Mining, ou Fouille De Données (FDD), n'est qu'une étape du processus global d'Extraction de Connaissances à partir de Données. En effet, l'ECD se réfère à une démarche complète d'exploitation des données intégrant leur préparation pour permettre l'application des algorithmes de Data Mining suivie de la validation des modèles obtenus [Kod 1967]. Le Data Mining se situe, donc, au cœur de ce processus.

Nous débutons ce chapitre par des définitions détaillées des concepts de Data Mining et d'ECD. Par la suite, nous expliquons quelques facteurs intéressants qui sont à l'origine de l'émergence de domaine de l'ECD. Puis, nous citons rapidement les différents types de données qui peuvent être manipulées par les techniques d'extraction de connaissances (ou de Data Mining). Après une description détaillée de processus de l'ECD (section 1. 5), nous nous focalisons sur la phase d'extraction de connaissances proprement dite, qui est le Data Mining, en rappelant les différentes tâches de Data Mining, ainsi que les différentes techniques utilisées pour les résoudre. Enfin, nous présentons brièvement les supports de stockage les plus communs pour les algorithmes de Data Mining.

1. 2 Définitions

Inventée au premier workshop de *KDD* en 1989 [Pia 1991], l'expression « Extraction de Connaissances à partir de Données » désigne l'ensemble de « *processus non trivial d'identification des modèles valides, nouveaux, potentiellement utiles et finalement compréhensibles à partir des données d'une grande base de données* » [Fay 1996a] [Han 2000]. En effet, les données représentent un ensemble de faits et le modèle l'expression, exprimée dans un langage de description d'un ensemble de données, qui décrit les corrélations entre ces données [Fay 1996b].

Le terme *processus* implique que l'ECD comporte plusieurs étapes, qui englobe la collection de données, la préparation de données, la recherche de modèles, et enfin l'interprétation et l'évaluation des modèles ainsi obtenus [Han 2000] [Fay 1996a] [Bra 1996], tout répétées dans des itérations multiples. La qualification *non triviale* signifie que les calculs effectués au niveau de chaque étape du processus de l'ECD sont des calculs complexes qui nécessitent plusieurs opérations de recherche et d'induction. En plus, les modèles découverts par le processus de l'ECD devraient être *valides* sur des nouvelles données, sous une forme *compréhensible* par les utilisateurs, et enfin ces modèles doivent apporter quelque chose de *nouveau* et de potentiellement *utile* afin que l'utilisateur puisse en bénéficier.

La finalité de processus de l'ECD est d'aboutir à des nouvelles connaissances valides et potentiellement utiles. En effet, la connaissance n'est qu'un modèle suffisamment intéressant et sûr [Fay 1996b]. L'exactitude d'un modèle est estimée par des mesures, appelées « *mesures de certitude* » ou « *critères de certitude* », spécifiées par les utilisateurs ou les experts de domaine. Les mesures de certitude [Sil 1995], combinées avec la validité, la nouveauté, l'utilité et la simplicité des modèles, permettent à l'utilisateur de garder seulement les modèles interprétables en connaissances utiles et de débarrasser des autres qui n'apportent pas d'intérêt. Ainsi, en résulte du processus de l'ECD un ensemble de modèles pas tous interprétables en connaissances exploitables par les utilisateurs.

Les concepts Data Mining et ECD sont parfois confondus et considérés comme synonymes. Cependant, la définition la plus partagée du concept de Data Mining le voit comme une étape essentielle d'un processus global d'Extraction de Connaissances à partir de Données [Fay 1996a] [Fay 1996b]. Cette étape consiste en « *application des méthodes et techniques d'exploration de données, spécifiques au Data Mining, afin d'en tirer des nouvelles connaissances sous forme de modèles présentés à l'utilisateur averti pour examen* » [Gar 2000]. En effet, le Data Mining fait appel à un lot de méthodes issues de la statistique, de l'analyse de données, de l'intelligence artificielle ou de l'apprentissage automatique ; les plus populaires de ces méthodes sont détaillées dans le chapitre 3.

1. 3 Les facteurs d'émergence de l'ECD

La croissance extrêmement rapide des données collectées dans les bases de données, et la nécessité d'une réactivité efficace de la part des décideurs face à ces informations nouvelles ont stimulés cette dernière décennie le développement rapide de l'Extraction de Connaissances à partir de Données. Elle a été définie par ses précurseurs comme « *l'extraction, à partir des données, d'une information implicite, inconnue auparavant et potentiellement utile* » [Fay 1996a]. Cette caractérisation a été revisitée par différents auteurs mais tous s'accordent sur le fait qu'il s'agit d'extraire des connaissances pertinentes et intelligibles d'une grande masse de données. Les facteurs les plus importants, en plus de la croissance des volumes de données, qui ont poussés les chercheurs à accélérer les travaux dans le domaine de l'ECD sont :

Trop de données disponibles et peu de connaissances

Beaucoup de données sont rassemblées depuis les premières bases de données documentaires (1957). Cependant, qu'a-t-on appris de toutes ces données ? Quelles connaissances pouvons-nous tirer de toutes ces données ?

Au début de 1984, dans son Megatrends [Nai 1986], John Naisbitt a observé que « *nous nous noyons dans les renseignements mais nous sommes affamés de connaissances* ». Le problème aujourd'hui ce n'est pas l'absence de données, mais plutôt il n'y a pas assez d'analystes humains disponibles, compétents et synthétiques pouvant traduire toutes ces données en connaissances utiles.

L'émergence des entrepôts de données

Les outils de Data Mining regroupent des nouveaux produits du domaine de l'aide à la décision. Ces produits résultent des super bases de données, appelées entrepôts de données (*Data Warehouses*), qui contiennent l'ensemble des informations d'un organisme sous forme harmonisée et accessible. Dans les entreprises centralisées, l'expert consacrait beaucoup de temps à extraire les connaissances et les analyser suite à une interrogation d'une base de données. De nos jours, une simple requête d'une base de données peut renvoyer des milliers d'enregistrements à l'expert, cela est dû à la croissance effrénée des volumes d'informations. L'expert doit cependant répondre d'une manière efficace pour satisfaire aux contraintes qui lui sont imposées.

Des machines de hautes performances sont accessibles

L'application des techniques de Data Mining sur des très grandes bases de données distribuées engendre un coût de calcul inacceptable, impose ainsi le recours à des traitements parallèles et distribués. En effet, l'émergence des réseaux informatiques et des machines parallèles de haute performances a beaucoup stimulée les chercheurs de domaine de l'ECD à paralléliser et à distribuer les techniques de Data Mining, puisque l'exploitation de parallélisme améliore considérablement leurs temps de réponse sur les grands volumes de données. Ainsi, une nouvelle branche dans le domaine de l'ECD a été révélée, l'Extraction Parallèle et Distribuée de Connaissances (EPDC) [Fu 2001] [Par 2003] [Guo 1999] [Zak 2000], ou encore « *Parallel and Distributed Knowledge Discovery (PDKD) »* en anglais.

1. 4 Les différents types de données

Les données manipulées par les algorithmes de Data Mining ne sont que les valeurs des champs des enregistrements des tables de la base de données, ou de l'entrepôt de données. Ces données possèdent un type qu'il est important de préciser. En effet, la plupart des méthodes sont sensibles à la nature des données manipulées. Par exemple, certaines méthodes exigent la présence des données *quantitatives* alors que d'autres peuvent être sensibles à la présence des données *qualitatives*.

Données quantitatives

Une donnée quantitative prend ces valeurs dans l'ensemble des entiers ou réelles, elle dite alors *discrète* (par exemple, l'âge, le nombre d'enfants, etc.) ou *continue* (par exemple, le salaire, le prix unitaire, etc.).

Données qualitatives

Par définition, une donnée qualitative n'est pas numérique, mais prend ces valeurs dans un ensemble fini dont les éléments correspondent à des caractéristiques des objets, appelées *modalités*. Ces valeurs sont le plus souvent alphanumériques ; elles peuvent être numérisées par des codes distincts, mais non comme des quantités arithmétiques de plein statut. On distingue généralement deux classes de données qualitatives, selon que les modalités sont ordonnées (données qualitatives *ordinale*) ou pas ordonnées (données qualitatives *nominale*).

1. 5 Le Processus de L'ECD

Comme elle montre la figure 1. 1, l'Extraction de Connaissances à partir de Données est un processus constitué de plusieurs étapes [Bra 1996] [Fay 1996c] [Fay 1996d] [Fay 1996e] ;

elles sont répétées dans des itérations multiples (des feedbacks et des boucles récursives peuvent être observés durant le processus) et à chaque itération ou étape du ce processus une intégration des connaissances des expertes de domaine est nécessaire (le processus est en perpétuelle interaction avec les utilisateurs) pour découvrir de nouvelles connaissances cachées interprétables et utilisables. De ce fait, le processus de l'ECD est souvent qualifié d'itératif et d'interactif [Bra 1996]. Les étapes de ce processus consistent principalement en collection des données contenant dans les différentes sources opérationnelles de l'entreprise, la préparation des données nécessaires pour accomplir la ou les tâches de Data Mining souhaitées, l'application des méthodes de Data Mining nécessaires pour résoudre ces tâches et enfin l'évaluation et la validation des résultats obtenus.

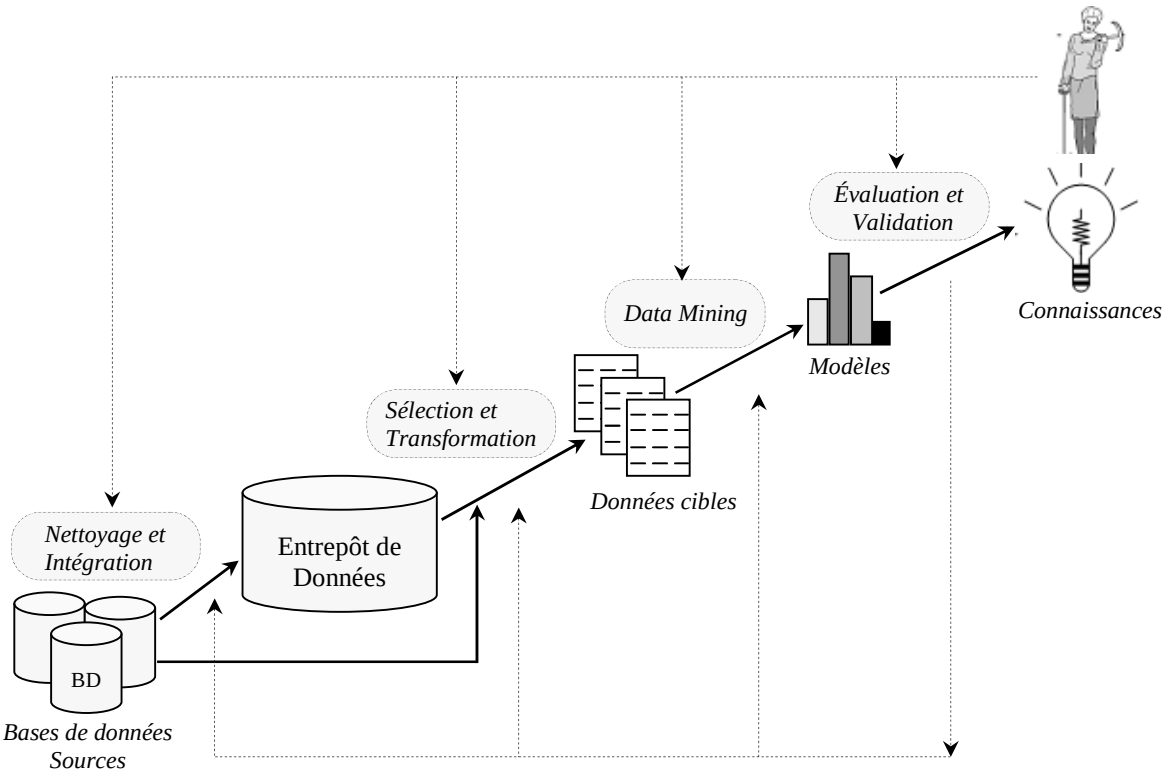


Figure 1. 1 Étapes du processus d'Extraction de Connaissances à partir de Données

1. 5. 1 La collection de données

Cette étape consiste généralement à collecter les données contenant dans les différentes sources opérationnelles de l'entreprise, éventuellement réparties et hétérogènes, après nettoyage, prétraitement, transformation et consolidation de ces données, dans une base de données cohérente ou le plus souvent dans un entrepôt de données. La construction d'un entrepôt de données, qui fait l'objet du chapitre 2, est pratiquement réalisée au niveau de cette étape. Dans cette section, nous expliquons seulement les tâches de nettoyage et d'intégration de données, effectuées lors de construction de l'entrepôt de données, puisque elles ont un

impact important sur les étapes ultérieures du processus de l'ECD et ensuite sur les modèles résultants.

1. 5. 1. 1 Nettoyage de données

Puisque l'entrepôt de données est construit pour fin d'extraire des connaissances utiles pour la prise de décisions stratégiques, il est important que les données collectées dans l'entrepôt soient correctes. Cependant, puisque les grands volumes de données des différentes sources opérationnelles de l'entreprise sont impliqués, il y a une haute probabilité d'erreurs et d'anomalies dans les données. Donc, la routine de nettoyage de donnes [Fam 1997] [Han 2000] consiste à éliminer, ou à réduire le maximum possible, les données erronées, aberrantes, inconsistantes, bruyantes et les valeurs manquantes dans l'ensemble de données collectées avant de les chargées dans l'entrepôt de données (voir figure 1. 2).

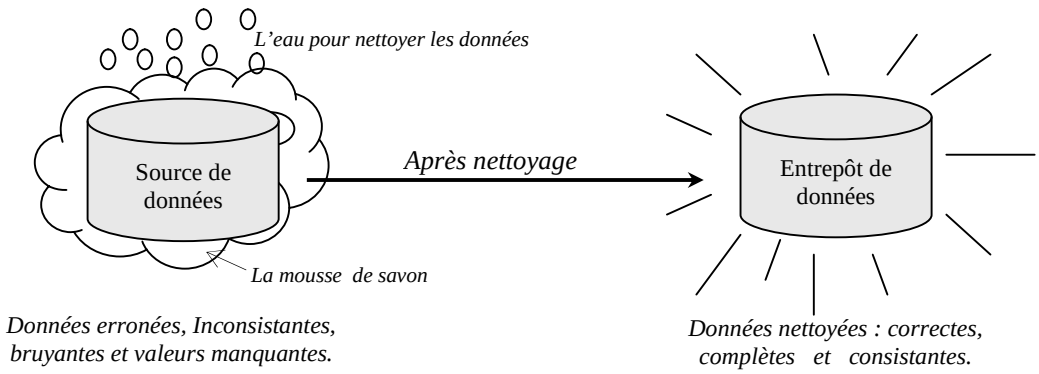


Figure 1. 2 Routine de nettoyage de données

Il existe plusieurs méthodes de nettoyage de données. En effet, [Han 2000] présentent d'utiliser des techniques comme la régression linéaire et le Clustering pour le traitement des donnes bruyantes. Quant au traitement des données manquantes, [Han 2000] présentent un ensemble d'opérations à savoir :

- Ignorer les enregistrements contenant des valeurs manquantes ;
- Remplir les valeurs manquantes manuellement ;
- Employez une constante globale pour remplir les valeurs manquantes ;
- Utiliser une valeur moyenne pour remplir les valeurs manquantes ;
- Utiliser la valeur la plus probable pour remplir les valeurs manquantes (exemple de techniques : la formule de Bayes ou l'arbre de décision) ;
- Chercher à estimer ces valeurs manquantes par des méthodes d'induction comme la régression, les réseaux de neurones ou les graphes d'induction.

1. 5. 1. 2 Intégration de données

L'intégration de données [Cal 2001] implique la combinaison de multiples sources de données, éventuellement réparties et hétérogènes, dans une base de données cohérente qui sera adaptée par la suite pour l'extraction de connaissances. Comme elle montre la figure 1. 3, les sources de données peuvent inclure plusieurs bases de données relationnelles, les bases de données de production (transactionnelles), les bases de données multidimensionnelles, les fichiers plats ou même d'autres sources externes à l'entreprise. La combinaison de telles sources de données construite un entrepôt de masse volumes de données et plus important de données homogènes, résumées, consolidées qui facilite la tâche d'analyse et d'exploration de données. En effet, le but de construction d'un entrepôt de données est la création des vues consolidées pour aider le processus d'extraction de connaissances et ensuite celui de la prise de décision.

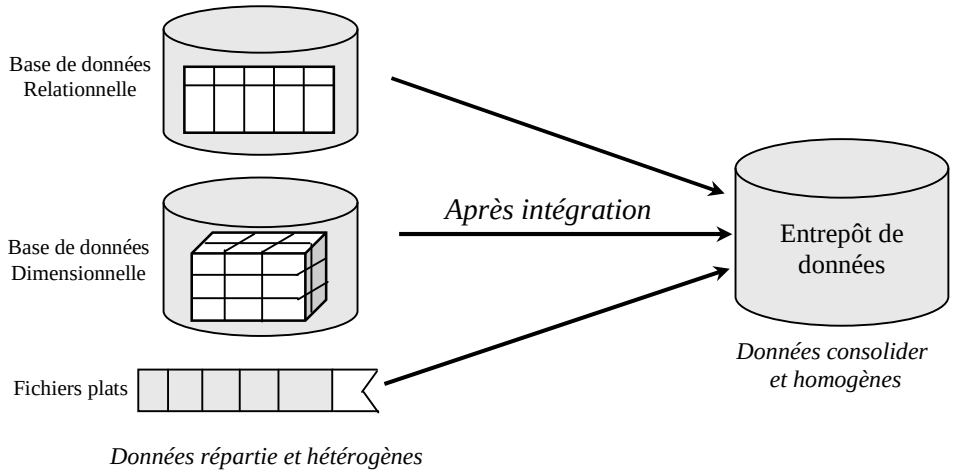


Figure 1. 3 Phase d'intégration de données

Il existe plusieurs problèmes à considérer durant la phase d'intégration de données. En effet, l'intégration des schémas des sources fait apparaître des conflits, depuis longtemps bien répertoriés dans la littérature. Les principaux conflits pouvant survenir entre deux schémas sont les suivants :

- *Les problèmes de terminologie* : Un conflit de terminologie survient lorsqu'un même objet du réel est désigné par des noms différents ou au contraire lorsqu'un même nom est utilisé pour deux objets différents. Ces cas peuvent correspondre à des problèmes de synonymie ou d'homonymie, mais sont le plus souvent dus à une différence de niveau de généricité (par exemple : « Personne » et « Étudiant »), ou à des converses (par exemple : « Vente » et « Achat »). En outre, les conflits de terminologie sont souvent à l'origine des problèmes de la redondance des données ; elle constitue un autre problème important à résoudre lors de l'intégration des données et sa détection fait souvent référence à la méthode d'analyse des corrélations [Han 2000].

- *Les incompatibilités de contraintes* : Un conflit de contraintes apparaît lorsque sur deux concepts établis comme équivalents ont des contraintes incompatibles (par exemple : « Un âge supérieur à 18 » et « Un âge inférieur à 17 »).
- *Les conflits de structures* : Les conflits de structure sont caractérisés par un choix différent de propriétés à stocker pour un même concept du réel. Par exemple, on peut définir une personne dans une vue par son numéro, son nom et son âge, et dans une autre vue par son nom, son prénom et son adresse.
- *Les conflits de représentation* : On détecte un conflit de représentation lorsque deux représentations différentes sont choisies pour les mêmes propriétés d'un même concept. Par exemple la date de commande peut être incluse dans la commande ou former un objet relié à la commande.

1. 5. 2 La Préparation de données

La qualité des résultats d'un processus de l'ECD dépend en grande partie de la qualité des données utilisées, d'où l'importance de l'étape de préparation de ces données [Fay 1996e]. D'après [Fam 1997], le prétraitement de données consiste en toute action effectuée sur les données avant l'application d'une technique de Data Mining. La collection des données dans un entrepôt de données facilite considérablement l'étape de préparation de données, puisque les données sont déjà nettoyées, transformées, fusionnées, agrégées, harmonisées et intégrées en vue de les préparées pour un processus d'aide à la décision. Il est important de distinguer entre la préparation de données pour le Data Mining et la préparation de données pour la conception d'un entrepôt de données. En effet, l'entrepôt de données permet de regrouper les données qui seront éventuellement accessibles aux méthodes de Data Mining, mais cet entrepôt ne supporte pas que le Data Mining. Il peut aussi supporter d'autres utilisations, comme l'analyse statistique et des applications de production qui nécessitent des données analytiques (moyennes, médianes, etc.). D'ailleurs, la préparation des données pour un entrepôt de données doit permettre d'optimiser les utilisations pour améliorer les temps de calculs lors de l'analyse. Par contre, la préparation des données pour le Data Mining doit être faite en fonction d'un ou plusieurs buts (ou tâches) définis à l'avance. C'est pourquoi on peut dire que la préparation des données pour le Data Mining est très différente de la préparation des données pour un entrepôt de données [Pyl 1999]. Malgré ces différences, les deux se complémentent très bien, et un entrepôt de données constitue une base solide pour supporter les techniques de Data Mining.

Les principaux traitements, effectués lors de l'étape de préparation de données, peuvent être résumés en sélection des données pertinentes pour accomplir la ou les tâches de Data Mining requises et ensuite, la transformation des données sélectionnées en une forme plus approprie pour réaliser ces tâches.

1. 5. 2. 1 Sélection de données

L'objectif de cette étape est de sélectionner et d'analyser l'état des données requises pour exécuter la ou les tâches de data Mining souhaitées. Il s'agit d'identifier les attributs (colonnes) et/ ou les enregistrements (lignes) à utiliser pour le Data Mining. Les critères de sélection incluent la pertinence de ces données, la qualité et les contraintes techniques, comme les limites de volume de données ou les types de données à utiliser. Les raisons éventuelles de la sélection de données peuvent être : la réduction du temps de construction du modèle qui décroît avec le nombre d'attributs, l'optimisation de la qualité des modèles obtenus (en suppriment les attributs fortement corrélés) ou les contraintes d'outils utilisés (nombre d'enregistrements supportés). Les étapes ultérieures du processus de l'ECD s'appliquent exclusivement sur l'ensemble de données sélectionné dans cette étape, d'où son importance.

1. 5. 2. 2 Transformation de données

Cette phase consiste à transformer les données en une forme appropriée pour accomplir la ou les tâches de Data Mining requises. Précisément, les transformations incluent la normalisation des valeurs des champs des enregistrements et ainsi que la réduction de nombre de champs des enregistrements des tables de la base de données, puisque chaque tâche de Data Mining se concentre seulement sur un sous ensemble de champs. Également, certaines d'autres modifications et combinaisons des champs des enregistrements peuvent être faites afin d'arranger les données originales dans un espace de données plus approprié aux tâches de Data Mining, qui seront exécutées à l'étape suivante (Data Mining).

La technique de normalisation de données consiste à appliquer des opérations de normalisation en vue de représenter les données sous une forme de mesures de la même plage de données. Par exemple, les données « -2, 23, 100, 59, 48 » peuvent être normalisées en « -0.02, 0.23, 1. 00, 0.59, 0.48 ».

La technique de réduction de données [Han 2000], ou « *Data Reduction* », peut être appliquée sur un ensemble de données pour obtenir une représentation réduite de celle-ci (voir la figure 1. 4). L'objectif de la réduction de données est d'avoir un espace de travail mois volumineux, au lieu de travail sur l'intégrité des données originale de très large volume. Puisque l'extraction de connaissances à partir d'un ensemble de données réduite doit être plus efficace et produit le même (ou presque le même) résultat analytique, il est important de travail sur des ensembles de données réduits. Parmi les stratégies de réduction de données, nous citons l'agrégation par les cubes de données, la compression de données, la discrétisation et la génération de hiérarchie [Han 2000].

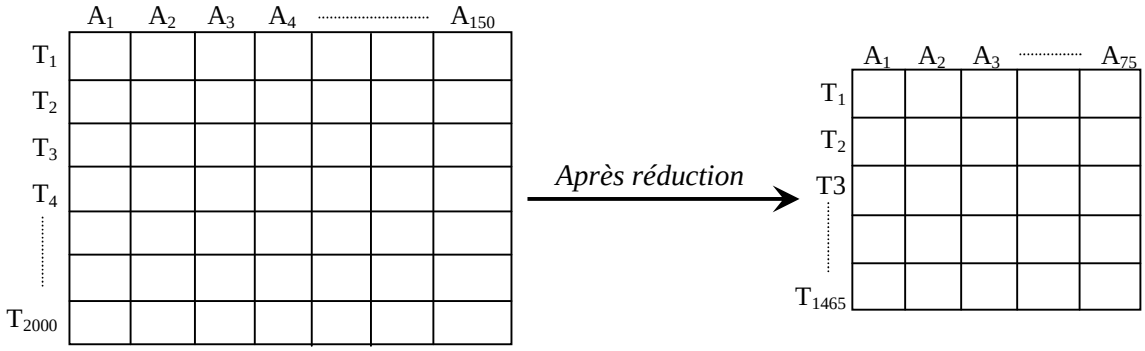


Figure 1. 4 Réduction de données

1. 5. 3 Le Data Mining

L'étape de Data Mining constitue le cœur du processus d'Extraction de Connaissances à partir de Données. Elle s'agit d'appliquer des méthodes intelligentes, spécifiques au Data Mining, afin d'extraire à partir de données, dite données d'apprentissage, des modèles, des règles ou toutes autres formes compréhensibles et interprétables en connaissances utiles (voir la figure 1. 5). Parmi les méthodes les plus connues du Data Mining, nous citons les méthodes de classification, les méthodes de Clustering et les méthodes de recherche de règles d'association. Dans le chapitre 3, un état de l'art sur ces différentes méthodes est réalisé. En générale, durant cette étape, elle est souhaitable de mettre en œuvre différentes techniques ou méthodes de Data Mining afin de les comparer et d'en retenir une ou plusieurs combinées [Gar 2000].

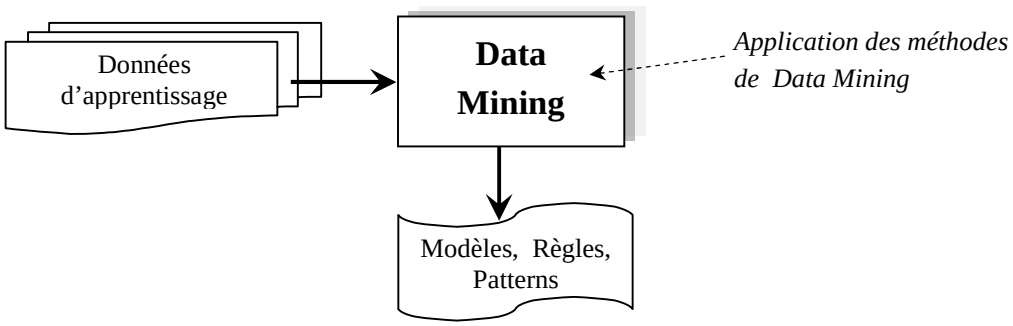


Figure 1. 5 Étape Data Mining du processus de l'ECD

1. 5. 4 L'évaluation et validation

C'est l'étape finale du processus d'Extraction de Connaissances à partir de Données. Elle consiste à évaluer les résultats obtenus pour déterminer quels modèles peuvent être considérés comme des connaissances nouvelles et intéressantes. Cette étape comporte aussi une interprétation des résultats et une comparaison des modèles. En effet, la pertinence de la connaissance découverte est estimée par des critères de certitude imposés par les utilisateurs

ou les experts de domaine [Sil 1995]. Ensuite, les modèles énumérés comme des connaissances pertinentes seront validés sur d'autres ensembles de données ou sur d'autres systèmes.

Les méthodes de validation vont dépendre de la nature de la tâche et du problème considéré. Par exemple, pour la segmentation et l'association, la validation est essentiellement du ressort de l'expert qui jugera de la pertinence des segments constitués (segmentation) ou de la pertinence des règles (association). Aussi, pour faire de la classification, on décompose les données en trois ensembles disjoints : un ensemble d'apprentissage, un ensemble de test et un ensemble de validation. Au mois deux ensembles sont nécessaires : l'ensemble d'apprentissage permet de générer le modèle et l'ensemble de test permet d'évaluer l'erreur réelle du modèle sur un ensemble indépendant. Ainsi, lorsqu'il s'agit de tester plusieurs modèles et de les comparer, on peut sélectionner le meilleur modèle selon ses performances sur l'ensemble de test et ensuite évaluer son erreur réelle sur l'ensemble de validation [Gil 2002]. Ce processus de validation est illustré sur la figure ci-dessous.

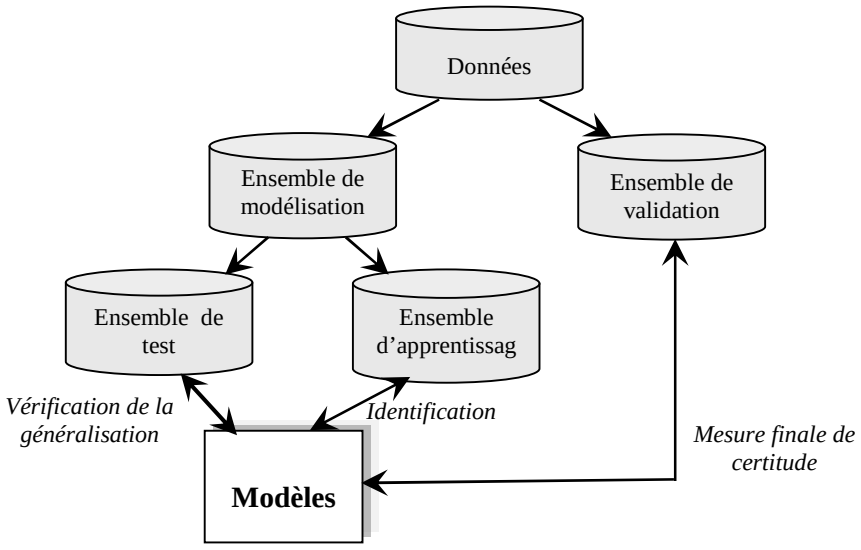


Figure 1. 6 Processus de validation

Il existe une variété de techniques de validation. Parmi les plus connues, nous citons la matrice de confusion et la validation croisée.

La matrice de confusion

La validation d'un modèle peut être mesurée par un comptage de nombres de cas corrects et incorrects. La technique de validation par matrice de confusion s'intéresse au rapport entre le nombre de cas corrects sur le nombre de cas total pour élaborer le modèle sur les cas non utilisés. En effet, cette technique consiste à élaborer une matrice à deux dimensions, dont l'élément (*i*, *j*) indique le nombre de cas de la classe Y^i prédits par le

modèle et observés comme étant réellement de la classe $\hat{Y}^j$ sur les données de validation (voir la figure 1. 7) [Gar 2000]. Ainsi, lorsque $i = j$, cette quantité représente le nombre correcte d'affectation.

		Classes d'affectation		
		$\hat{Y}^1$	…	$\hat{Y}^2$
Classes d'origines	Y^1	n_{11}	…	n_{1m}
	…	…	…	…
	Y^m	n_{m1}	…	n_{mm}

Figure 1. 7 Matrice de confusion

Alors, le taux de succès, noté *Succ*, pour *m* classes peut être calculé par l'équation suivante :

$$Succ = \frac{\sum_{i=1}^{m} n_{ii}}{\sum_{i=1}^{m}\sum_{j=1}^{m} n_{ij}}$$

Le taux d'erreur, noté *Err*, n'est autre que le complément à 1 du taux de succès :

$$Err = 1 - Succ$$

La validation croisée

Lorsqu'on ne dispose pas de suffisamment d'exemples, on peut se permettre d'apprendre et d'estimer les erreurs avec un même ensemble par la technique de validation croisée (voir la figure 1. 8). Celle-ci permet d'estimer l'erreur réelle d'un modèle selon l'algorithme suivant :

Validation croisée (*E*, *k*)
E est un ensemble de données et *k* est un entier.
Diviser *E* en *k* parties de tailles égales {E_1, …, E_k}
Pour *i* de *1* à *k*
Construire un modèle *M* en apprenant sur l'ensemble E-E_i
Mesure l'erreur e_i de *M* sur l'ensemble E_i
Fin Pour
Retourner l'erreur réelle *e* qui est donnée par la moyenne des erreurs mesurées e_i :

$$e = \frac{\sum_{i=1}^{k} e_i}{k}$$

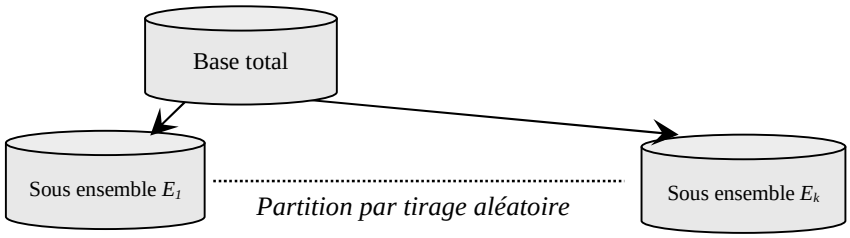


Figure 1. 8 Processus de validation croisée

1. 6 La topologie des tâches de Data Mining

Dans ce qui suit, nous considérerons quelques tâches traitées par le Data Mining [Tuf 2003] [Lef 1998] [Lar 2005]. Les techniques utilisées pour réaliser ces tâches sont nombreuses, et il arrive souvent qu'une technique propose des résultats exploitables pour l'une ou l'autre de ces tâches. D'après [che 1996a], les tâches de Data Mining peuvent être classifiées selon les critères suivants : (a) le type de la base de données sur laquelle on travail, (b) la connaissance à découvrir, (c) la technique de Data Mining à utiliser. Dans cette section, nous expliquons brièvement les différentes tâches de Data Mining, présentées suivant une classification selon deux critères : En fonction des objectifs des problèmes à résoudre, nous avons cinq tâches principales et en fonction du type d'apprentissage utilisé, nous avons deux types de tâches. La dernière partie de cette section est réservée à la description de types des techniques et méthodes utilisées pour la résolution de ces tâches.

1. 6. 1 Selon l'objectif

On fonction des objectifs des problèmes à résoudre, les tâches de Data Mining se partagent entre la classification, l'estimation, la prédiction, la segmentation et les règles d'associations.

1. 6. 1. 1 Classification

Elle consiste à apprendre à classer des objets dans des classes prédéfinies à partir d'exemples déjà classés. L'application de la classification construit un modèle à partir d'un ensemble d'apprentissage, où les éléments sont groupés dans des classes prédéfinies, et ensuite ce modèle sera utilisé pour classifier automatiquement des nouveaux objets dans l'une des classes prédéfinies (voir figure 1. 9).

Les techniques les plus appropriés à la classification sont : la méthode des *k*-Plus Proches Voisins, les arbres de décision, les réseaux bayésiens et les réseaux de neurones artificiels.

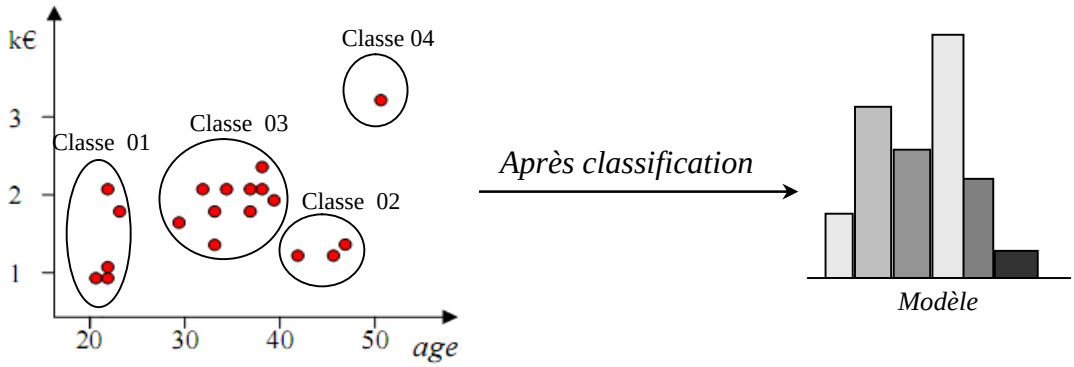


Figure 1. 9 Application de la classification

1. 6. 1. 2 Estimation

Elle consiste à estimer la valeur d'un champ à partir des caractéristiques d'un objet. Le champ à estimer est un champ à valeurs continues. L'estimation peut être utilisée dans un but de classification. Il suffit d'attribuer une classe particulière pour un intervalle de valeurs du champ estimé. Généralement, pour l'estimation des valeurs manquantes des champs des enregistrements, on utilise les réseaux de neurones artificiels, la régression linéaire simple et la régression linéaire multiple.

1. 6. 1. 3 Prédiction

L'objectif de la prédiction est d'estimer la valeur future pour certains attributs dans la base de données en se basant sur d'autres attributs. Cette tâche est similaire à la classification mise à part la nature de la variable cible. Tel que, dans la classification la variable cible est de type catégoriel tandis que dans la prédiction la variable cible est de type numérique. Les techniques utilisées pour la classification et l'estimation peuvent être utilisées pour le traitement des problèmes de prédiction.

1. 6. 1. 4 Segmentation

La segmentation, ou encore appelée « *Clustering* » en anglais, consiste à diviser un ensemble de données en classes (ou clusters), tel que pour une mesure de similarité donnée, les objets d'une classe soient les plus similaires possibles (homogénéité intra-classes) et que les objets des classes différentes soient le plus différentes possible (séparabilité inter-classes). A la différence de classification, les classes dans le cas de segmentation ne sont pas définies par l'analyste mais découvertes au cours de l'opération. On utilise souvent les algorithmes de type nuées dynamiques (les centres mobiles, *k*-means, etc.), la Classification Hiérarchique Ascendante (CHA) et les cartes auto-organisatrices de Kohonen pour résoudre les problèmes de segmentation.

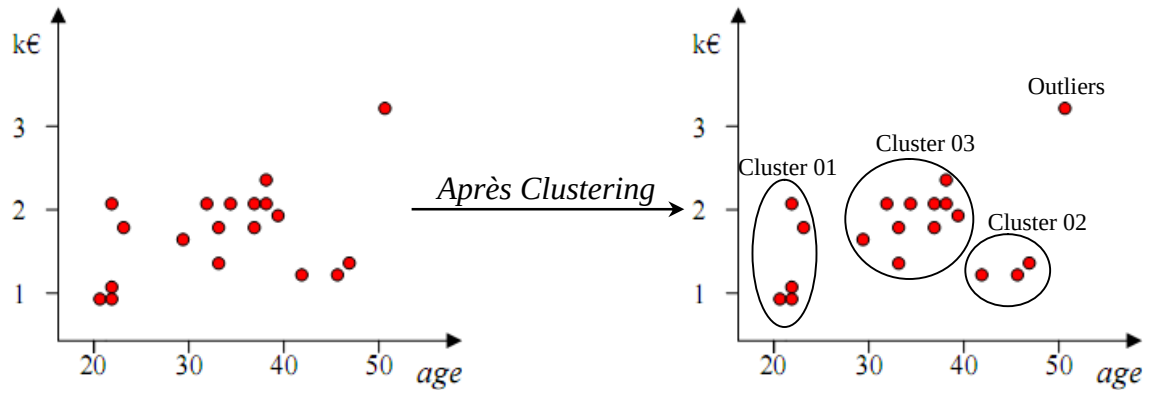


Figure 1. 10 Application de Clustering

1. 6. 1. 5 Règles d'association

Il s'agit de trouver les associations ou les relations intéressantes entre les éléments d'un ensemble de données. Ces associations sont exprimées sous forme d'une suite de règles simples et facilement compréhensibles par les utilisateurs (par exemple, les règles de forme *IF-THEN*). L'application principale de problème de recherche de règles d'association est « l'analyse du panier de la ménagère », c'est-à-dire la recherche d'associations entre produits sur les tickets de caisse. Le but de la méthode est l'étude de ce que les clients achètent pour obtenir des informations sur qui sont les clients et pourquoi ils font certains achats [Gil 2002]. La méthode cherche les produits qui pourront être achetés ensemble.

1. 6. 2 Selon le type d'apprentissage

On fonction de type d'apprentissage utilisé, nous avons :

1. 6. 2. 1 Apprentissage supervisé

Les méthodes à apprentissage supervisé ont pour objectif l'explication et/ ou la prédiction d'un ou plusieurs phénomènes observables et effectivement mesurés. Concrètement, elles vont s'intéresser à une ou plusieurs variables de la base de données définies comme étant les cibles de l'analyse. La classification, la prédiction et l'estimation sont des exemples de tâches à apprentissage supervisé.

1. 6. 2. 2 Apprentissage non supervisé

Les méthodes à apprentissage non supervisé permettent de travailler sur un ensemble de données dans lequel aucune des données ou des variables à disposition n'a d'importance particulière par rapport aux autres, c'est-à-dire un ensemble de données dans lequel aucune variable n'est considérée individuellement comme la cible, l'objectif de l'analyse. La

segmentation et la recherche de règles d'association sont des exemples de tâches à apprentissage non supervisé.

1. 6. 3 Le type de techniques de Data Mining

Les principaux algorithmes de Data Mining se répartissent en deux grandes familles de techniques :

1. 6. 3. 1 Techniques descriptives (exploratoires)

Les techniques descriptives, sans variables privilégiées, visent à mettre en évidence des informations présentent mais cachées par le volume de données. Ces techniques s'intéressent aux individus ainsi qu'aux groupes homogènes qu'ils peuvent former [Tuf 2003]. Elles sont destinées à la résolution des problèmes d'apprentissage non supervisé (segmentation et recherche de règles d'association).

1. 6. 3. 2 Techniques prédictives (explicatives)

Les techniques prédictives, avec variables à expliquer, visent à extrapoler de nouvelles informations à partir des informations présentes. Ces nouvelles informations pouvant être qualitatives (classification) ou quantitatives (prédiction). Ces techniques s'intéressent aux variables et aux relations entre elles [Tuf 2003]. Elles sont destinées à la résolution des problèmes d'apprentissage supervisé (classification, prédiction et estimation).

De nombreux algorithmes appartenant à ces deux familles ont été développés. Nous présenterons, dans le tableau ci-dessous, les plus répondus parmi eux.

Les techniques descriptives	**Les techniques prédictives**
✓ Les méthodes factorielles ; ✓ Les techniques de type nuées dynamiques (les centres mobiles, *k*-means, etc.) ; ✓ La Classification Hiérarchique Ascendante (CHA) ; ✓ Les cartes auto-organisatrices de Kohonen ; ✓ Les règles d'association.	✓ Les arbres de décision ; ✓ Les réseaux de neurones artificiels; ✓ L'analyse discriminante ; ✓ Les modèles de régression (la régression linéaire simple et multiple, la régression logistique) ; ✓ Les Machines à Support de Vecteur (MSV) ; ✓ Les réseaux bayésiens ; ✓ Le *k*-Plus Proches Voisins.

Tableau 1. 1 Principales techniques descriptives et prédictives

Il n'existe pas de meilleure technique de Data Mining, il faudra faire des compromis selon les besoins dégagés et les caractéristiques connues des outils. Par conséquent, à tout jeu de données et tout problème correspond une ou plusieurs méthodes. Le choix se fait en fonction [Gil 2002] :

- De la tâche à résoudre ;
- De la nature et de la disponibilité des données ;
- Des connaissances et des compétences disponibles ;
- De la finalité des modèles construits. Pour ce la les critères suivant sont importants : complexités de la construction du modèle, complexité de son utilisation et ses performances.

1. 7 Supports de stockage pour le Data Mining

Le Data Mining devrait être applicable à n'importe quel support de stockage de données [Han 2000]. En effet, les défis présentés par les algorithmes et les approches de Data Mining changent de manières significatives une fois appliquées à différents formats de stockage de données.

Les supports de stockage les plus communs pour les algorithmes de Data Mining sont : les fichiers plats, les bases de données transactionnelles, les bases de données relationnelles et les entrepôts de données.

1. 7. 1 Fichiers plats

Pendant la première décennie de l'émergence de Data Mining, Les fichiers plats constituent le support de stockage le plus commun pour les algorithmes de Data Mining, notamment pour le niveau recherche. Les fichiers plats sont des fichiers de données simple en format texte ou binaire avec une structure connue par les algorithmes de Data Mining à appliquer. Généralement, ces fichiers peuvent contenir des transactions, des mesures scientifiques, etc.

1. 7. 2 Bases de données transactionnelles

En général, une base de données transactionnelle est un fichier constitue d'un ensemble d'enregistrements, où chacun représente une transaction de la base de données. Chaque transaction est constituée d'un unique identificateur, appelé *TID* (*Transaction IDentification*), et d'un ensemble d'items (liste d'items). En outre, la base de données transactionnelle peut avoir des tables additionnelles associées à elle, qui contiennent des informations concernant

chaque transaction ou chaque liste d'items de la base de données. Ces informations peuvent être la date de la transaction, l'identificateur de client, etc.

1. 7. 3 Bases de données relationnelles

Brièvement, une base de données relationnelle consiste en collection de plusieurs tables de données, dont chacune est assignée à un nom unique. Chaque table de la base de données relationnelle comporte un ensemble d'attributs (colonnes ou champs) et stocke un grand nombre de tuples (enregistrements ou lignes). Chaque tuple d'une table relationnelle représente un objet ou un individu identifié par une clé unique et décrit par un ensemble des valeurs d'attributs. Donc, les données d'une base de données relationnelle sont représentées dans des tableaux à deux dimensions ($n \times P$). Pour fixer les idées, les lignes ($i=1,\ldots, n$) représentes les n individus et les colonnes ($j=1,\ldots, P$) sont alors les P attributs ou variables observées sur les individus.

	Attribut 1	*Attribut 2*	...	*Attribut j*	...	*Attribut P*
Individu 1	x_{11}	x_{12}	...	x_{1j}	...	x_{1p}
Individu 2	x_{21}	x_{22}	...	x_{2j}	...	x_{2p}
...	...	...	...	...	...	...
Individu i	x_{i1}	x_{i2}	...	x_{ij}	...	x_{ip}
…	...	...	...	...	...	...
Individu n	x_{n1}	x_{n2}	...	x_{nj}	...	x_{np}

Les données peuvent également être représentées par une matrice d'ordre (n, P). Son terme générique est X_{ij} ($j^{ème}$ attribut du $i^{ème}$ individu).

$$X = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1j} & \ldots & x_{1P} \\ x_{21} & x_{22} & \ldots & x_{2j} & \ldots & x_{2P} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i1} & x_{i2} & \ldots & x_{ij} & \ldots & x_{iP} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nj} & \ldots & x_{nP} \end{pmatrix}$$

1. 7. 4 Entrepôts de données

En vue d'effectuer des analyses pour le capital d'informations d'une entreprise, les différentes bases de données de l'entreprise sont intégrées dans une grande base de données centralisée et cohérente, ou encore dans un « *entrepôt de données* ». En effet, l'entrepôt de données contient des données consolidées et agrégées pendant une longue période de temps, et utilisées pour faciliter les tâches de l'analyse de données et ainsi que celles de Data Mining. Généralement, l'entrepôt de données est modélisé par une structure multidimensionnelle

(tableau multidimensionnelle, cube de données ou hypercube) dans laquelle chaque dimension correspond à un attribut ou à un ensemble d'attributs des tables relationnelles et chaque cellule stocke la valeur d'une certaines mesures agrégées. La figure suivante montre une vue à trois dimensions (cube de données) : Locations, Éléments et Temps.

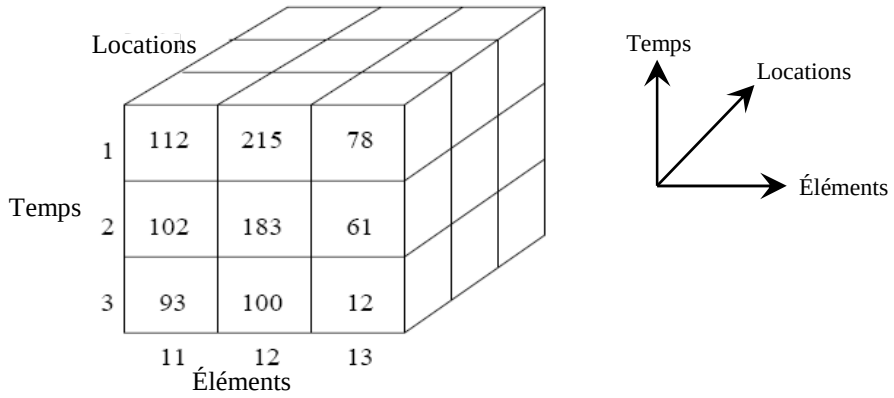


Figure 1. 11 Vue multidimensionnelle de données (cube de données)

Une fois les données sont transformées et prétraitées dans une forme approprie pour l'exploration et l'analyse de données et collectées dans un entrepôt, les algorithmes de Data Mining peuvent être exécutés directement sur cet entrepôt de données dont la modélisation est souvent relationnelle (schéma en étoile, schéma en flocon de neige, etc.). Donc, comme dans le cas des bases de données relationnelles, l'entrée d'un algorithme de Data Mining est une simple table plate comportant un ensemble d'attributs (ou colonnes) et un ensemble d'enregistrements (ou lignes). En général, chaque ligne représente un individu et les colonnes les propriétés des individus. Une telle table représente la forme d'une matrice qu'est traditionnellement utilisées dans les statistiques (voir section 1. 7. 3)

1. 8 Conclusion

Actuellement, La plupart des recherches dans le domaine de l'ECD assument que les données sont statiques. Dans la pratique, ces données sont maintenues dans des entrepôts de données, qui constituent le support de stockage le plus adapté pour les applications d'analyse et d'exploration de données, telles que le Data Mining. La construction d'un entrepôt de données est souvent considérée comme une étape d'un processus global d'Extraction de Connaissances à partir de Données. Cependant, certains auteurs le voir comme un projet à part indépendant de processus de l'ECD, puisque sa construction est une tâche très complexe et nécessite plusieurs années pour la finaliser.

Pour bien comprendre le concept d'entrepôt de données, un état de l'art sur ce concept est réalisé dans le chapitre suivant.

CHAPITRE 2

État de l'art sur les Entrepôts de Données

2. 1 Introduction

Le concept d'entrepôt de données a pris forme au début des années 1990, il est devenu depuis la clé de voûte de ce que l'on appelle l'informatique décisionnelle. La vogue grandissante de ce concept est la résultante de l'évolution de l'informatique dans les organisations. Il s'agit en fait de mettre à la disposition des gestionnaires tout ou une partie des données qui ont été accumulées dans les fichiers et les bases de données opérationnelles de l'entreprise depuis de nombreuses années et qui ne peuvent être exploitées en efficace. L'acquisition de ces données a coûté cher aux entreprises et leurs gestionnaires sont de plus en plus conscients de ce qu'ils perdent en n'y ayant pas accès pour prendre leurs décisions. Malheureusement ces nombreuses données acquises sont généralement difficilement accessibles pour des multiples raisons : elles sont dispersées, disparates, incohérentes, mal connues, et leurs mises à jour n'est pas synchronisées. Leur format sur les supports informatiques répond aux exigences des systèmes transactionnels pour lesquels ont été définies. Comme le prix des mémoires à accès direct ainsi que des postes de travail connectés à des bases de données ont considérablement chutés, entraînant dès lors l'apparition sur le marché de nombreux outils de manipulation de données, il est devenu techniquement et économiquement possible de recopier et de réorganiser de grandes masses de données opérationnelles à des fin uniquement décisionnelles : c'est le domaine des entrepôts de données [Pas 1998].

Pour bien comprendre le concept d'entrepôt de données, nous débutons ce chapitre par une comparaison entre les systèmes opérationnels (ou OLTP) et les systèmes d'entrepôts de données (ou OLAP). À la suite de cette comparaison, nous rappelons les différentes approches d'intégration de données, puisque l'entrepôt n'est qu'une approche d'intégration de plusieurs sources de données hétérogènes et réparties. Dans la section 2. 4, nous présentons une définition précise pour le concept d'entrepôt de données suivie d'une description détaillée de l'architecture des systèmes d'entrepôts (section 2. 5). Dans le reste de chapitre, nous nous focalisons sur la conception et l'exploitation des entrepôts de données en présentant, dans cette ordre, les différentes stratégies de conception et de construction des entrepôts de

données (section 2. 6), la modélisation de données adoptée aux entrepôts de données (section 2. 7), la manipulation des données de l'entrepôt (section 2. 8) et enfin, son processus d'alimentation (section 2. 9).

2. 2 Systèmes OLTP versus systèmes OLAP

La tâche principale des systèmes de bases de données opérationnelles, appelés aussi les systèmes OLTP (*On-Line Transaction Processing*), est d'exécuter les transactions en ligne et de traiter les requêtes. Ces systèmes couvrent la plupart des opérations quotidiennes d'une organisation, telles que l'achat, la fabrication, les opérations bancaires, etc. Les systèmes d'entrepôts de données, d'autre part, aident les décideurs dans leurs rôles de l'analyse de données et de la prise de décision. Tels systèmes peuvent organiser et présenter les données dans divers formats afin de satisfaire les besoins divers des différents utilisateurs. Ces systèmes sont connus en tant que les systèmes OLAP (*On-Line Analytical Processing*).

Les applications OLTP automatisent les tâches de traitement de données telles que la saisie des commandes et les transactions. Ces tâches sont structurées et répétitives, et consistent des transactions courtes, atomiques et isolées. Les transactions exigent des données détaillées et mises à jour, et lisent ou mettent à jour quelques enregistrements (des dizaines) accédés sur leurs clés primaires. Les bases de données opérationnelles adoptent un modèle de données Entité/Relation et une conception de la base orientée application, ainsi que leurs capacité de stockage varie entre des centaines de méga-octets aux giga-octets. La consistance et la récupération de la base données sont critiques, et maximiser le débit des transactions est la métrique clé de performance.

Les entrepôts de données, en revanche, sont visés pour le support d'aide à la décision. Ils contiennent des données historisées, agrégées, résumées et consolidées qui sont les plus adaptées pour réaliser les activités d'analyse exigées par les décideurs. Ainsi, pour faciliter le processus de la prise de décision, qui nécessite l'analyse et la visualisation des données stockées dans l'entrepôt, ce dernier adopte une modélisation multidimensionnelle de données et une conception de la base orientée sujet. Puisque les entrepôts de données contiennent des données consolidées de plusieurs bases de données opérationnelles, pendant des périodes potentiellement longues, ils ont tendance à être d'ordre de grandeur plus grand que les bases de données opérationnelles ; les entrepôts de données ont une capacité de stockage varie entre des centaines de giga-octets à des téra-octets. Les charges de travail sont des requêtes intensives avec la plupart sont ad hoc, ainsi que des requêtes complexes qui peuvent accéder à des millions d'enregistrements et d'effectuer un grand nombre d'opérations de balayage de données, de jointures et d'agrégations. Le débit des requêtes et le temps de réponse sont les plus importants que le débit des transactions.

Le tableau ci-dessous résume ces différences entre les systèmes OLTP et les systèmes OLAP, [Han 2000].

Caractéristiques	**OLTP**	**OLAP**
Objectifs	Gestion et production	Consultation et analyse
Utilisateurs	Administrateurs et gestionnaires de BDD	Travailleurs intellectuels, analystes et décideurs
La conception de la base de données	Basée sur le modèle E/R, orientée application	Basée sur le modèle multidimensionnel, orientée sujet
Données	Courantes, garanties et mises à jour	Historiques, agrégées, maintient en plus le temps
L'unité de travail	Petite, simple transaction	Requête complexe
Les accès	Lecture/ Écriture	Lecture seulement
Opérations	Indexation, le hachage sur la clé primaire	Beaucoup d'opérations de Balayage de données, de jointures et d'agrégations
Nombre d'enregistrements accessibles	Dizaines	Millions
Nombre d'utilisateurs	Milliers	Centaines
Taille de la BDD	100 Mo à Go	100 Go à To
Métrique	Débit des transactions	Débit des requêtes et temps de réponse

Tableau 2. 1 Différences entre les systèmes OLTP et les systèmes OLAP

2. 3 Les approches d'intégration de données

Un entrepôt de données constitue avant tout une alternative pour l'intégration de diverses sources de données. Un système d'intégration a pour objectif d'assurer à un utilisateur un accès à des sources multiples, réparties et hétérogènes, à travers une interface unique. En effet, l'avantage d'un tel système est que l'utilisateur se préoccupe davantage de ce qu'il veut obtenir plutôt que comment l'obtenir, l'objectif étant l'obtention d'informations. Ainsi, cela le dispense des tâches telles que chercher et trouver les sources de données adéquates, interroger chacune des sources de données en utilisant sa propre interface et combiner les différents résultats obtenus pour finalement disposer des informations recherchées. Différentes solutions ont été proposées face au problème de l'hétérogénéité des sources réparties de données. En effet, pour faire des recherches sur l'ensemble de ces sources, une intégration de celles-ci est nécessaire. Deux approches sont alors envisageables : migrer les requêtes vers les sources de données ou migrer les données pour les centraliser dans une source cible. Ceci consiste à suivre respectivement une approche « *non matérialisée* », appelée aussi approche de médiation ou une approche « *matérialisée* », appelée aussi approche d'entreposage [Bou 2003].

2. 3. 1 Approche de médiation

La figure 2. 1 montre une architecture pour une approche d'intégration des sources de données à base de médiation. Elle consiste à définir une interface entre l'agent (humain ou logiciel) qui pose une requête et l'ensemble des sources. Cette interface donne l'impression d'interroger un système centralisé et homogène alors que les sources interrogées sont réparties, autonomes et hétérogènes. L'approche de médiation présente l'intérêt de pouvoir construire un système d'interrogation de sources de données sans toucher aux données qui restent stockées dans leurs sources d'origine. Ainsi, il n'y a pas d'intégration de données, autrement dit pas de centralisation de ces dernières.

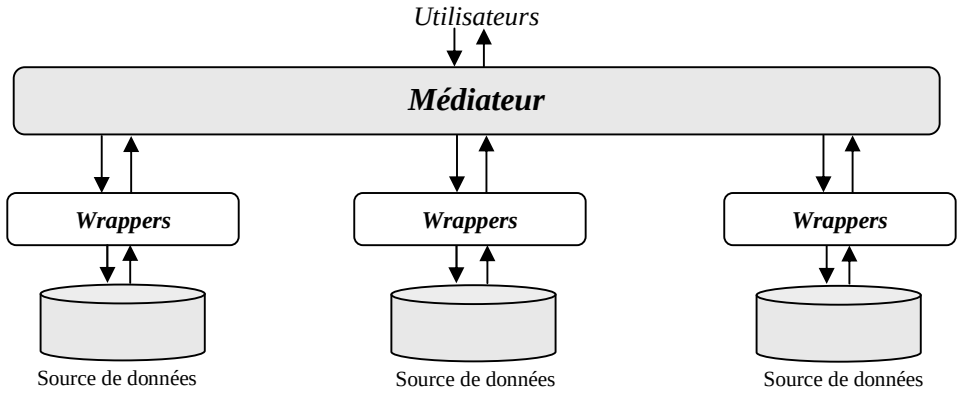


Figure 2. 1 Architecture de l'approche de médiation

Quand une requête est posée à un site client, le médiateur (ou l'intégrateur) s'occupe de la traduction de celle-ci en requêtes appropriées pour les différents sites hétérogènes impliqués [Wie 1992]. Ces requêtes sont alors mappées et envoyées aux processeurs de requêtes locales. Les résultats retournés à partir des différents sites, interrogés par des adaptateurs (ou des wrappeurs), sont intégrés en jeu global de réponse. Cette approche à base d'interrogations exige des processus complexes de filtrage et d'intégration de l'information, qui sont en concurrence pour l'accès aux ressources de traitement aux niveaux des sources locales. Elle est inefficace et potentiellement chère pour les requêtes fréquentes, notamment pour les requêtes exigeant des agrégations.

2. 3. 2 Approche d'entreposage

C'est une autre approche intéressante qui remplace l'approche traditionnelle d'intégration des sources de données hétérogènes décrite ci-dessus. Plutôt que d'utiliser l'approche à base d'interrogations, l'entreposage de données utilise une approche à base de mise à jour, dans laquelle les informations en provenance de plusieurs sources de données hétérogènes sont intégrées à l'avance et stockées dans un entrepôt pour l'interrogation et l'analyse directes (voir la figure 2. 2 [Wid 1995]). À la différence des bases de données

opérationnelles, les entrepôts de données ne contiennent pas l'information le plus récente. Cependant, l'entrepôt apporte des performances élevés pour l'intégration des systèmes hétérogènes puisque les données sont copiées, prétraitées, intégrées, agrégées et restructurées dans une mémoire de données sémantique. En plus, le traitement des requêtes dans l'entrepôt de données n'interfère pas avec le traitement dans les sources locales. D'ailleurs, les entrepôts de données peuvent stocker et intégrer les informations historiques et supporter des requêtes multidimensionnelles complexes.

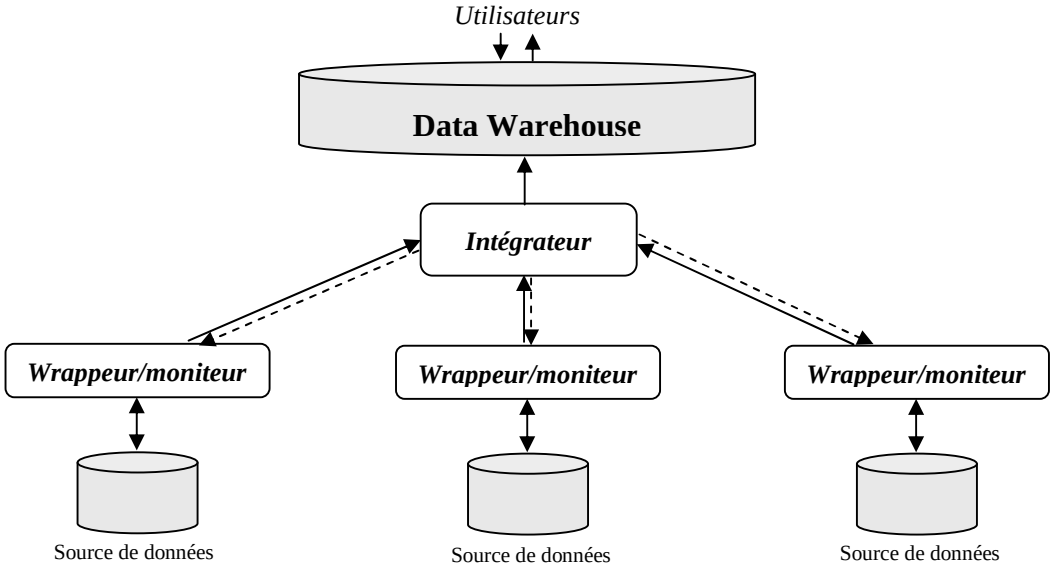


Figure 2. 2 Architecture de l'approche d'entreposage

2. 4 Définition

Un entrepôt de données, appelé aussi « *Data Warehouse* » en anglais, est un lieu de stockage intermédiaire de différentes données en vue de constitution d'un système d'information décisionnelle. D'après W. H. Inmon [Inm 1996], un entrepôt de données est définit comme suit :

> « *Un entrepôt est une collection de données orientées sujet, intégrées, non volatiles et historisées, organisées pour le support d'un processus d'aide à la décision* ».

Cette définition présente les caractéristiques principales d'un entrepôt de données. L'explication détaillée de ces caractéristiques est donnée dans la suite [Han 2000] [Cle 2001] [Mar 2006] :

Orientées sujet :

Les données de l'entrepôt sont organisées autour des sujets majeurs et des métiers de l'entreprise, tels que le client, le fournisseur, le produit et les ventes. L'avantage de cette

organisation en sujet demeure dans le fait qu'il devient possible de réaliser des analyses sur différents sujets de l'entreprise, et aussi, de faire des analyses par itération, sujet après sujet.

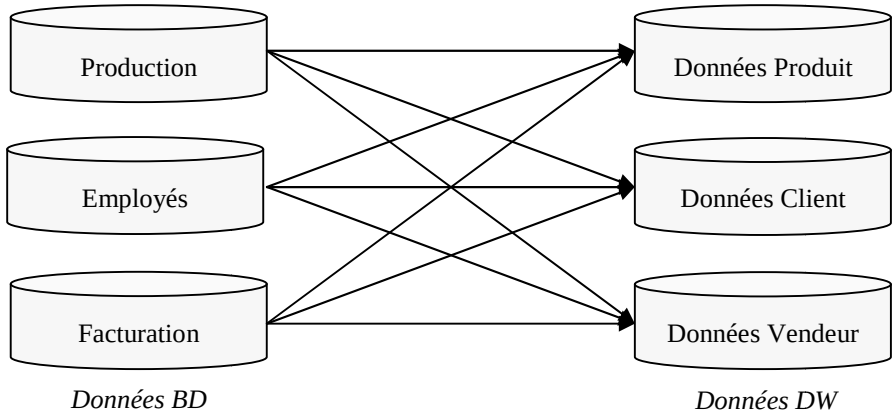


Figure 2. 3 Données Orientées sujet

Intégrées

Ce qui caractérise peut-être le plus un entrepôt de données est le fait qu'il collecte et l'intègre des données en provenances de multiples sources réparties et hétérogènes, telles que les bases de données relationnelles, les bases de données transactionnelles, les fichiers plats, etc. Avant d'être intégrées dans l'entrepôt, les données doivent être mises en forme et unifiées afin de correspondre au standard défini dans l'entrepôt de données.

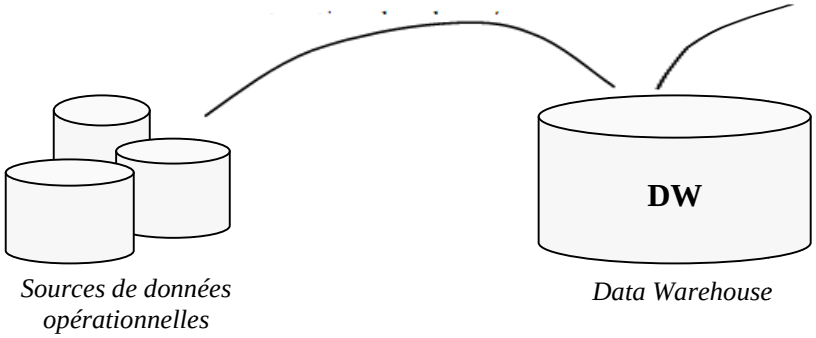


Figure 2. 4 Données intégrées

Non volatiles

Un entrepôt de données exige seulement deux opérations d'accès aux données : le chargement initial de données et l'accès aux données. Donc, la non volatilité implique la durabilité des données : les données dans l'entrepôt ne doivent ni être modifiées ni supprimées.

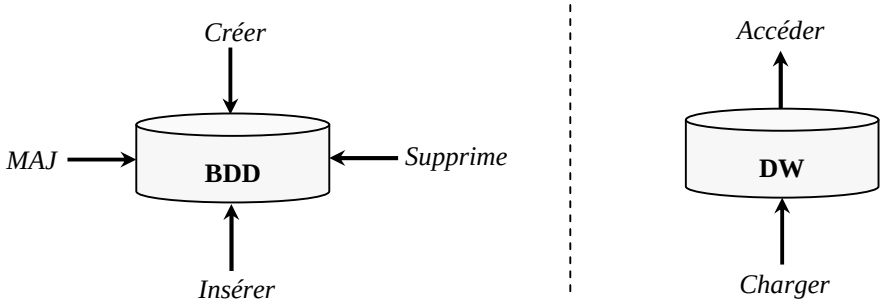


Figure 2. 5 Données non volatiles

Historisées

Les données sont stockées pour fournir les informations d'une perspective historique (par exemple, les 5 ou 10 dernières années). Chaque structure principale dans l'entrepôt de données contient, implicitement ou explicitement, un élément de temps.

Enfin, un entrepôt de données a un objectif et un groupe d'utilisateurs spécifiques, il doit constituer une base d'informations sur laquelle les décideurs et les gestionnaires de l'entreprise se basent pour prendre des décisions. Il s'agit dans la plupart de temps des décisions d'ordre tactiques et/ou stratégiques. Puisque l'entrepôt de données est destiné pour aider à la prise de décision, il n'est pas nécessaire d'y stocker toutes les données. Seules celles susceptibles de contribuer à l'amélioration des décisions prises suffisantes.

Cependant, le terme entreposage de données, ou « *Data Warehousing* » en anglais, se réfère souvent au processus de construction et d'utilisation de l'entrepôt de données [Han 2000], [Bal 1998]. La construction d'un entrepôt de données exige l'intégration, le nettoyage et la consolidation des données, alors que son utilisation nécessite une collection de technologies d'aide à la décision, telles que les outils OLAP, les outils de Data Mining, etc. Ces technologies permettent aux décideurs d'avoir une vue d'ensemble des données de l'entrepôt afin de faciliter leurs tâches, la prise de décisions.

2. 5 L'architecture d'un entrepôt de données

L'architecture simplifiée d'un entrepôt de données, considérée dans la plupart des travaux de recherche [Cha 1997] [Kim 2000] [Han 2000] est celle montrée dans la figure 2. 6. Les différents composants de cette architecture ont été intégrés dans plusieurs parties : les sources de données, la zone de préparation de données, l'entrepôt de données, les serveurs de présentation de données et les outils d'exploration et d'analyse de données. On parle généralement d'architectures n-tiers en raison des différentes couches possibles pour gérer les données et réaliser des analyses.

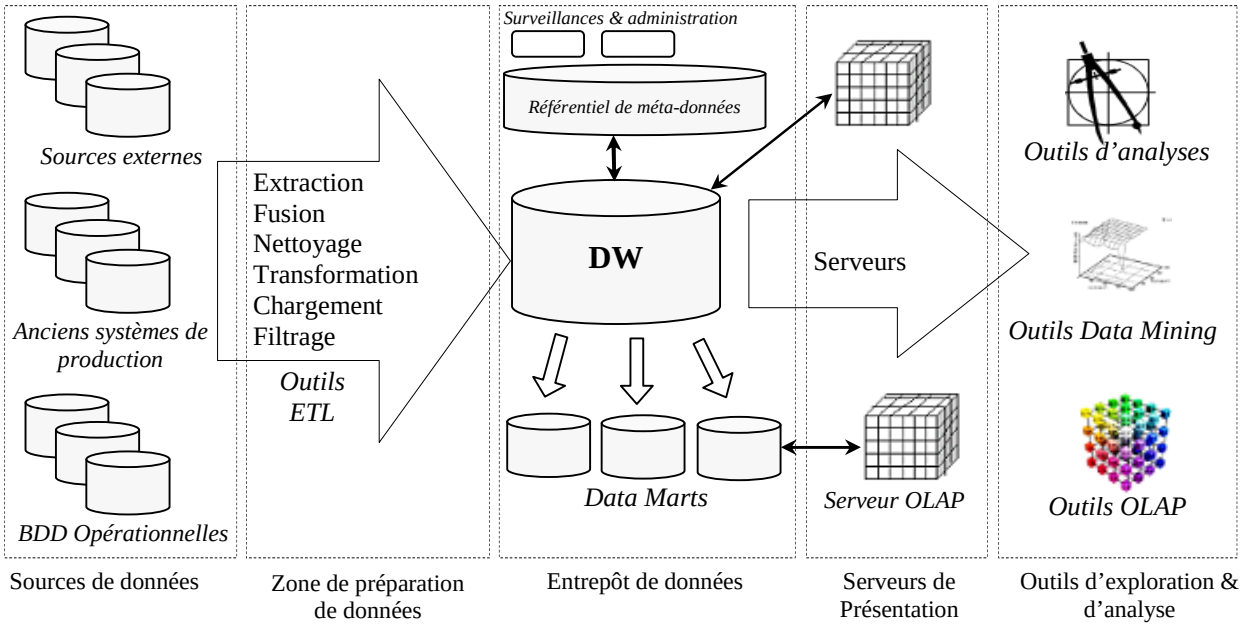


Figure 2. 6 Architecture d'un entrepôt de données

2. 5. 1 Les sources de données

Ce sont les bases de données sources à partir desquelles l'entrepôt de données est peuplé. Ainsi, les données de l'entrepôt sont de nature diverses ; elles proviennent de plusieurs sources de données réparties et hétérogènes, qui peuvent être internes ou externes à l'entreprise. Les sources de données internes correspondent à l'ensemble des systèmes opérationnels de l'entreprise, ainsi que des anciens systèmes de production qui contiennent des données encore exploitées par l'entreprise. Tandis que, les sources de données externes représentent des données externes à l'entreprise (Internet, bases de partenaires, etc.).

2. 5. 2 La zone de préparation de données

Cette zone englobe l'ensemble de processus qui permet d'extraire, de nettoyer, de transformer, de fusionner et d'archiver les données dans l'entrepôt. Les outils ETL (*Extract-Transform-Load*) ont en charge ces fonctions essentielles, qui permettre la préparation des données sources en vue de leurs intégration puis de leurs exploitation au sein du l'entrepôt de données [Mar 2006] [Cha 1997]. Donc, pour alimenter l'entrepôt de données, il faut établir des flux de données entre l'entrepôt et les différentes sources de données. Pour ce faire, il faut mettre en place des interfaces d'extraction, de transformation et d'alimentation entre les bases de productions et l'entrepôt de données. Cette phase d'alimentation de l'entrepôt, qui englobe les différentes phases de préparation de données, est détaillée dans la section 2. 9.

2. 5. 3 L'entrepôt de données

Il correspond au lieu de stockage massif et centralisé des données utiles pour les décideurs. Il est associé à un ensemble de méta-données (informations sur les données) qui forme en quelque sorte un référentiel de données contenant des informations permettant d'assurer différentes tâches telles que la maintenance de l'entrepôt de données. Ce référentiel est appelé « *référentiel de méta-données* » [Han 2000]; il contient toutes les informations concernant l'environnement de l'entrepôt telles que les modèles de données, toutes les informations concernant les sources de données et ainsi que toutes les règles nécessaires pour nettoyer, transformer, agréger et maintenir l'environnement de l'entrepôt.

Comme elle montre la figure 2. 7, les données stockées dans l'entrepôt peuvent être structurées en quatre classes de données, organisées selon un axe historique et un axe de synthèse [Bal 1998] [Tri 2005].

Données détaillées

Elles correspondent aux données de l'entrepôt qui n'ont pas subies aucune opération d'agrégation en vue de modifier leur niveau de détail. Elles peuvent être des données détaillées récentes ou anciennes, qui reflètent respectivement les événements les plus récents ou passés des systèmes opérationnels.

Données résumées ou agrégées

Ces données correspondent à un résultat d'analyse et une synthèse de l'information contenue dans l'entrepôt. Cette information synthétisée est obtenue par l'application des opérations d'agrégation de données qui permettre de réduire le niveau de détail des données de l'entrepôt. Donc, ces données sont moins détaillées que les premières et elles permettent de réduire le volume de données à stocker. Le type de données, en fonction de leur niveau de détail, permet de les classifier comme des données légèrement ou fortement résumées.

Données consolidées

Ce sont des données détaillées qui ont été nettoyées, ajustées, harmonisées et réunies pour fournir une source de données uniforme et de qualité exploitable par les analystes de données. L'objectif de la consolidation des données est de garantir la cohérence et la consistance de la base de l'entrepôt. En outre, en peut créer et maintenir des données historiques en consolidant les données dans l'entrepôt. Ainsi, nous pouvons dire que les données consolidées sont un type spécial de données dérivées des systèmes opérationnels [Bal 1998].

Données historisées

Chaque nouvelle insertion de données provenant du système opérationnel ne détruit pas les anciennes valeurs, mais créée une nouvelle occurrence de la donnée.

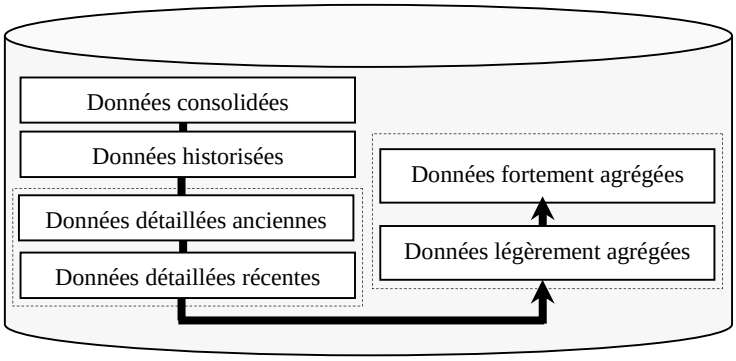


Figure 2. 7 Architecture de données dans un entrepôt de données

De point de vue de l'architecture, il y a quatre modèles d'entrepôt : l'entrepôt de l'entreprise, les Data Marts, l'entrepôt virtuel et l'ODS (*Operational Data Store*).

Entrepôt de l'entreprise

Il collecte toutes les informations concernant les sujets couvrant l'ensemble de l'entreprise. Il contient des données détaillées aussi bien que des données agrégées, et leur capacité de stockage peut dépasser les centaines de téra-octets. Ainsi, la conception et la construction d'un entrepôt pour l'ensemble de l'entreprise est une tâche très difficile à mettre en œuvre et nécessite beaucoup de temps pour la finaliser.

Data Marts

Le Data Mart, appelé aussi « *magasin de données* », est un sous ensemble logique de l'entrepôt de données qui contient des données spécifiques pour certaines sujets d'analyse, ou un processus métier, et ou un département particulier de l'entreprise. Le Data Mart contient un faible volume de données, ce qui rend la navigation et la recherche de données plus rapides. En effet, l'objectif de la répartition des données de l'entrepôt dans des Data Marts demeure dans l'augmentation des performances des requêtes complexes.

Entrepôt virtuel

C'est un ensemble de vues matérialisées[1] au dessus des bases de données opérationnelles. Pour assurer l'efficacité de processeur de requêtes, seulement certaines vues

[1] Une vue est une requête nommée. Une vue matérialisée est une table contenant les résultats d'une requête.

récapitulées peuvent être matérialisées. L'entrepôt de données se construit facilement mais exige des capacités extrêmes sur les serveurs de bases de données opérationnelles.

ODS : Operational Data Store

L'ODS est une structure intermédiaire qui stocke les données issues des systèmes opérationnels dans un format proche de ces derniers [Cle 2001]. C'est un stockage tampon avant la transformation et l'intégration des données dans l'entrepôt de données proprement dit. En effet, l'ODS constitue le point central à partir duquel on peut commencer à construire l'entrepôt de données.

D'après W.H. Inmon [Inm 1993], l'ODS est défini comme suit : « Une collection de données orientées sujet, intégrées, volatiles, courantes et détaillées pour supporter les décisions opérationnels et quotidiennes d'une organisation ».

2. 5. 4 Les serveurs de présentation de données

Un serveur de présentation est une machine cible sur laquelle l'entrepôt de données est stocké et organisé pour répondre en accès direct aux requêtes émises par les utilisateurs, les générateurs d'états et les autres applications. Selon la technologie de stockage de données (bases de données relationnelles, bases de données multidimensionnelles) et les techniques de traitement de ces données (requêtes SQL, les opérations OLAP), ces serveurs peuvent être répartis en trois classes de serveurs : les serveurs ROLAP (*Relational OLAP*), les serveurs MOLAP (*Multidimensional OLAP*) et les serveurs HOLAP (*Hybrid OLAP*) [Han 2000] [Tri 2005].

Serveurs ROLAP

Dans ce type de serveurs, les données sont stockées dans des bases de données relationnelles. Ils représentent une interface multidimensionnelle pour un SGBD relationnel. Le moteur OLAP est un élément supplémentaire qui fournit une vision multidimensionnelle de l'entrepôt, des calculs de données dérivées et des agrégations à différents niveaux. Il est aussi responsable de la génération des requêtes SQL mieux adaptées au schéma relationnel, qui profitent des vues matérialisées existantes pour exécuter efficacement ces requêtes. La figure 2. 8 montre une architecture d'un serveur ROLAP.

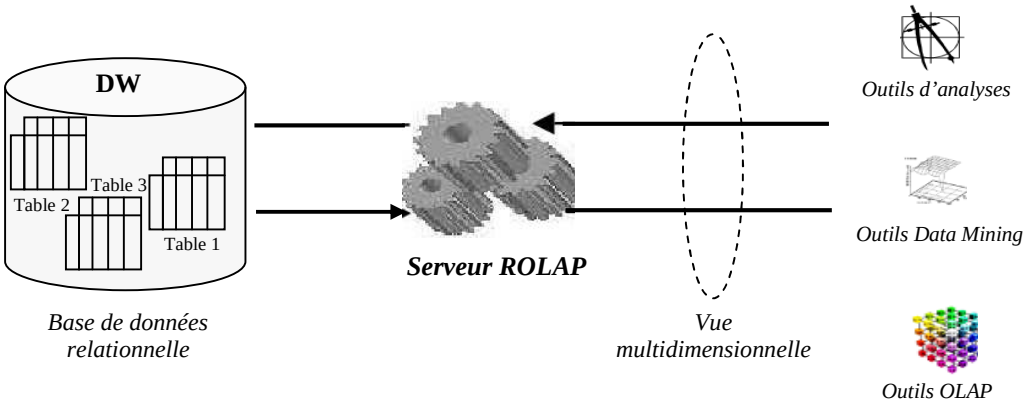


Figure 2. 8 Architecture d'un serveur ROLAP

Les serveurs ROLAP peuvent stocker de grands volumes de données, mais ils peuvent présenter un temps de réponse élevé. Les principaux avantages de cette catégorie de serveurs sont : (1) une facilité d'intégration dans les SGBDs relationnels existants, (2) une bonne efficacité pour stocker les données multidimensionnelles.

Serveurs MOLAP

Ce sont des serveurs qui stockent les données multidimensionnelles directement dans des structures de données spécialisées, appelées tableaux multidimensionnelles, et implémentent les opérations OLAP sur ces structures spéciales. Chaque dimension de ce tableau est associée à une dimension du cube de données (voir la section 2. 7). Seules les valeurs de données correspondant aux mesures de chaque cellule sont stockées. Les serveurs MOLAP demandent un pré-calcul de toutes les agrégations possibles. En conséquence, ils sont plus performants que les systèmes traditionnels, mais difficiles à mettre à jour et à gérer. La figure 2. 9 montre une architecture d'un serveur MOLAP.

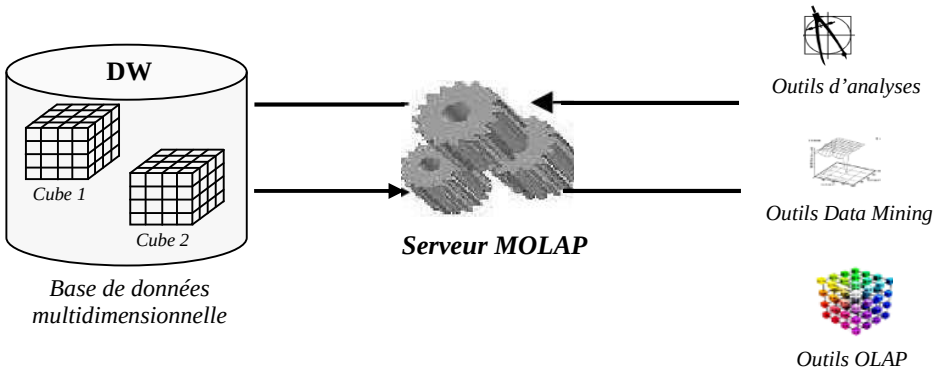


Figure 2. 9 Architecture d'un serveur MOLAP

Les serveurs MOLAP apparaissent comme une solution acceptable pour le stockage et l'analyse de l'entrepôt lorsque la quantité estimée des données de l'entrepôt ne dépasse pas

quelques giga-octets et lorsque le modèle multidimensionnel évolue peu. Mais, lorsque les données sont éparses, ces serveurs sont consommateurs d'espace [Cha 1997] et des techniques de compression doivent être utilisées.

Serveurs HOLAP

Ils supportent et intègrent un stockage de données multidimensionnel et relationnel d'une manière équivalente pour profiter des avantages des deux technologies ROLAP et MOLAP. Actuellement, la plupart des systèmes commerciaux utilisent une approche hybride, dont l'architecture est montrée sur la figure 2. 10. Cette approche permet de manipuler les informations de l'entrepôt de données avec un moteur ROLAP, tandis que pour la gestion des Data Marts, ils utilisent l'approche multidimensionnelle (MOLAP).

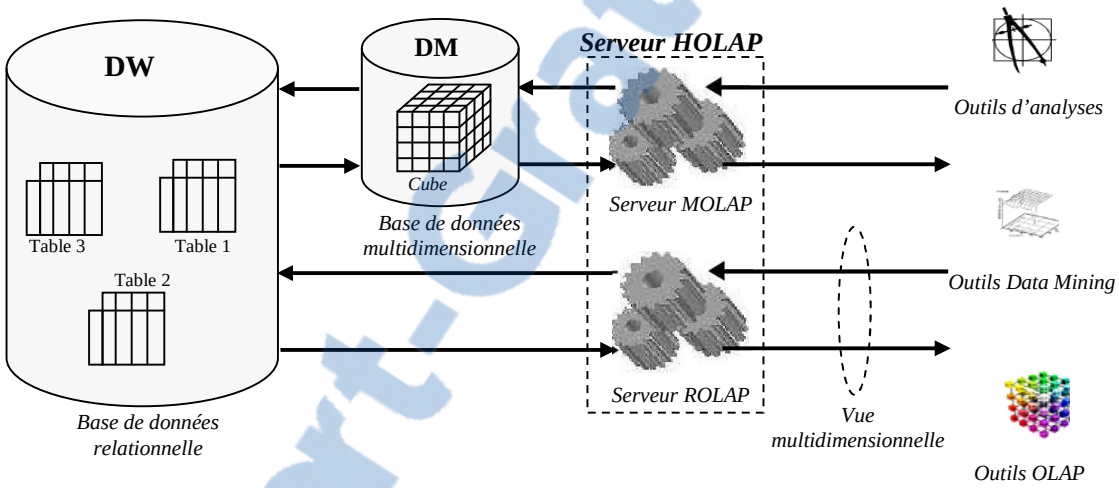


Figure 2. 10 Architecture d'un serveur HOLAP

2. 5. 5 Les outils d'exploration et d'analyse de données

Finalement, un entrepôt de données est destiné à être consulté et, donc, à aider les décideurs de l'entreprise dans leurs tâches de la prise de décisions. Il existe différents outils pour accéder aux informations contenant dans l'entrepôt de données [Cha 1997]. Parmi ces outils nous distinguons les outils d'analyse statistique, les outils de création de rapports (utilisés pour générer des tableaux de bord conventionnels), les outils d'interrogation (pour faciliter l'accès aux données en fournissant une interface conviviale au langage de requêtes), les outils OLAP (pour la synthèse et l'analyse des données multidimensionnelles) et les outils de Data Mining (pour la découverte des nouvelles connaissances).

2. 6 Les stratégies de conception et de construction des entrepôts de données

La construction et la mise en œuvre d'un entrepôt de données représentent une tâche complexe qui se compose de plusieurs étapes. La première consiste à l'analyse des sources de données et à l'identification des besoins des utilisateurs (l'étude préalable [Kim 2000]). La deuxième correspond à l'organisation des données à l'intérieur de l'entrepôt de données (l'étude de modèle de données de l'entrepôt). La troisième consiste à déterminer quelles données seront chargées, quelles transformations et vérifications seront nécessaires, la périodicité et le moment auxquels les transferts auront lieu (l'étude de l'alimentation). Finalement, la quatrième consiste à établir divers outils d'interrogation de l'entrepôt, tels que les outils de Data Mining, les outils d'analyse OLAP, etc. (l'utilisation et l'exploitation de l'entrepôt). Durant ce chapitre, nous expliquons en détail chaque étape à l'exception de l'étude préalable puisqu'il ressemble à toute étape préliminaire à l'implantation d'un nouveau système d'information automatisé. Cependant, dans cette section, nous présentons brièvement quelles approches de conception et de construction des entrepôts de données.

D'après [Han 2000], la conception des entrepôts de données doit être considérée selon ces quatre vues suivantes :

- *Vue Top-Down* : Elle permet de sélectionner les informations pertinentes pour les stockées dans l'entrepôt de données. Ces informations s'accordent aux besoins actuels et futurs de l'entreprise.
- *Vue source de données* : Elle expose les informations capturées, stockées et gérées par les systèmes opérationnels. Ces informations peuvent être documentées aux différents niveaux de détail et d'exactitude, à partir des tables des sources de données aux tables des sources de données intégrées. En effet, les sources de données sont souvent modélisées par un modèle relationnel classique (Entité/Relation).
- *Vue entrepôt de données* : Elle inclut les tables de faits et les tables de dimensions. Elle représente les informations stockées dans l'entrepôt de données. Cette vue inclut également les pré-calculs des totaux et des comptes, ainsi que les informations concernant les sources, les dates et les temps d'origine ajoutées pour fournir le contexte historique pour l'entrepôt de données.
- *Vue orientée requêtes* : C'est la perspective des données de l'entrepôt à partir d'un point de vue de l'utilisateur final. En effet, dans cette vue l'entrepôt est conçus à partir des besoins et des vues des utilisateurs finaux.

L'un des points clés de processus de construction et de conception de l'entrepôt de données réside dans la conception du schéma. En effet, la construction du schéma de l'entrepôt n'étant pas une tâche facile, plusieurs travaux ont proposés l'automatisation

partielle [Sou 2005] ou complète de cette tâche [Phi 2002] [Kim 2003]. Il n'existe à ce jour aucun consensus sur la méthodologie de conception de l'entrepôt, comme cela peut être le cas avec la méthode MERISE pour la conception des bases de données relationnelles. En conséquence, différentes pistes ont été proposées ; elles se basent sur l'utilisation de paradigmes variés tels que le paradigme Entité/Association [Try 1999], le paradigme objet grâce à la modélisation UML (*Unified Modeling Language*) [Luj 2005], etc.

De point de vue de conception de schéma de l'entrepôt de données, nous distinguons dans la littérature trois grandes approches [Fav 2007] : celle guidée par les données, qualifiée également d'ascendante ; celle guidée par les besoins d'analyse, dénommée également descendante et l'approche mixte qui combine les deux approches précédentes.

- *Approche orientée données* : Cette approche ignore les besoins d'analyse a priori. Elle consiste à construire le schéma de l'entrepôt à partir de ceux des sources de données et suppose que le schéma qui sera construit pourra répondre à tous les besoins d'analyse.
- *Approche orientée besoins d'analyse* : Elle propose de définir le schéma de l'entrepôt en fonction des besoins d'analyse et suppose que les données disponibles permettront la mise en œuvre d'un tel schéma. Parmi les approches orientées besoins d'analyse, on peut faire une distinction entre les approches guidées par les buts et les approches guidées par les utilisateurs.

  L'approche orientée buts suppose que le schéma de l'entrepôt est défini selon les objectifs d'analyse de l'entreprise. Ainsi, on suppose que tous les employés de l'entreprise ont des besoins d'analyses similaires vis-à-vis de l'exploitation de l'entrepôt. Autrement dit, tous les employés ont la même vision analytique de l'entrepôt de données. Tandis que, dans l'approche orientée utilisateurs, ces derniers sont interrogés afin de collecter l'ensemble de leurs besoins d'analyse avant de construire l'entrepôt, ce qui permet de garantir l'acceptation du système par les utilisateurs. Cependant, la durée de vie de ce schéma peut être courte, étant donné que le schéma dépend beaucoup des besoins des personnes impliquées dans le processus de développement de l'entrepôt.
- *Approche mixte* : Cette approche considère à la fois les besoins d'analyse et les données pour la construction du schéma. Ainsi, l'approche mixte est celle qui fait l'objet de plus d'investigations aujourd'hui. L'idée générale est de construire des schémas candidats à partir des données (démarche ascendante) et de les confronter aux schémas définis selon les besoins (démarche descendante) [Bon 2001] [Phi 2002] [Sou 2005]. Ainsi, le schéma construit constitue une réponse aux besoins réels d'analyse et il est également possible de le mettre en œuvre avec les sources de données.

De point de vue de l'architecture, la construction de l'entrepôt de données est réalisée selon plusieurs types d'architectures. Parmi ces architectures, nous distinguons l'architecture Entrepôt de Données centré, l'architecture Opérationnelle Data Store / Entrepôt de Données et l'architecture Data Marts / Entrepôt de Données.

- *Architecture Entrepôt de Données centré* : Elle correspond à l'architecture la plus simple pour les systèmes d'entrepôts de données (voir figure 2. 11). Dans cette architecture, les sources de données opérationnelles sont intégrées dans un entrepôt de données. Ce dernier contient des données détaillées aussi bien que des données agrégées, les plus adéquates pour satisfaire les besoins d'analyse des utilisateurs finaux de l'entrepôt de données.

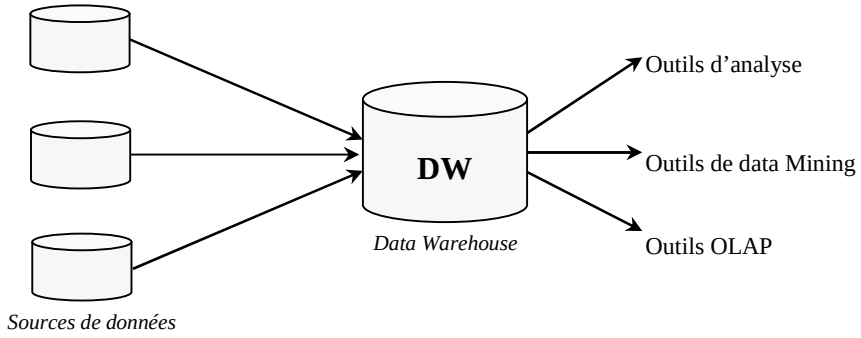


Figure 2. 11 Architecture Entrepôt de Données centré

- *Architecture Operational Data Store / Entrepôt de Données* : L'architecture ODS / Entrepôt de Données est une extension de l'architecture Entrepôt de Données centré [Inm 1993]. Dans cette architecture, les sources de données opérationnelles sont intégrées dans une base de données centralisée, appelée ODS (Operational Data Store). Ensuite, le contenue de l'ODS sera transférée dans l'entrepôt de données si nécessaire.

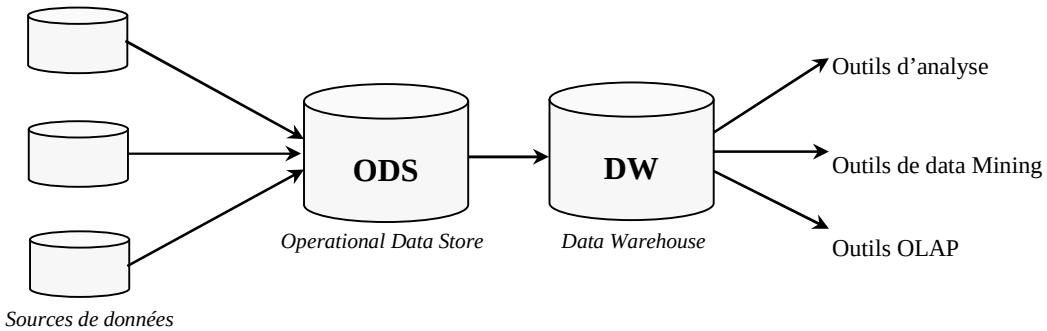


Figure 2. 12 Architecture Operational Data Store / Entrepôt de Données

- *Architecture Data Marts / Entrepôts de Données* : Dans cette architecture, la construction de l'entrepôt de données se fait selon deux approches : la première consiste à construire plusieurs Data Marts d'une manière isolée, un pour chaque métier ou département de l'entreprise, ensuite ces Data Marts seront intégrés pour construire l'entrepôt de données pour l'ensemble de l'entreprise (figure 2. 13 (a)). Tandis que la

deuxième approche consiste à construire, en premier lieu, un entrepôt global pour l'entreprise et ensuite les données de cet entrepôt seront distribuées pour remplir les Data Marts correspondant aux métiers ou aux départements de l'entreprise (figure 2. 13 (b)).

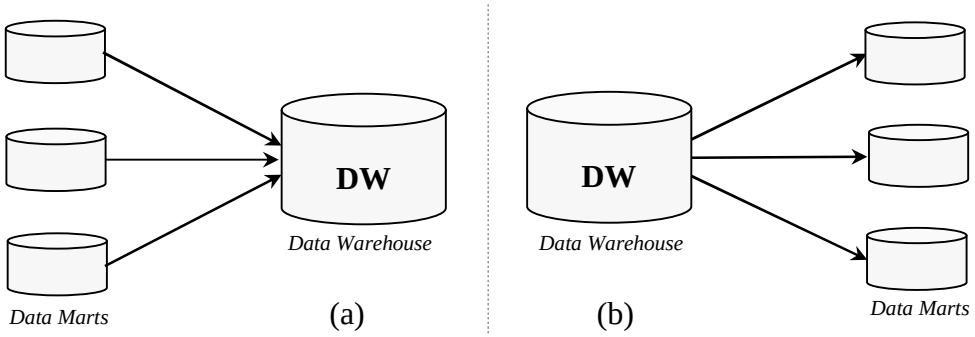


Figure 2. 13 Architecture Data Marts / Entrepôt de Donnée

Finalement, il n'y a pas une méthodologie précise pour la conception et la construction des entrepôts de données. En effet, Kimball [Kim 1996] a introduit le concept d'entrepôt de données, mais sans donner de méthodologie de construction, mis à part qu'il faut une base de données dénormalisée et qu'il est prudent d'oublier tout ce qu'on a appris en modélisation de base de données.

2. 7 La modélisation des entrepôts de données

Les données à analyser au niveau de l'entrepôt doivent refléter la vision d'une classe d'analystes [Kim 1996] [Mar 1998]. Cette vision correspond à une structuration des données de l'entrepôt selon plusieurs axes d'analyses (ou dimensions). En effet, l'organisation des données dans l'entrepôt se base sur deux concepts fondamentaux : le concept de fait qui représente le sujet d'analyse, et le concept de dimension qui spécifie la manière dont on regarde les données pour les analyser. Donc, un entrepôt de données présente une modélisation dite multidimensionnelle [Agr 1995a] [Agr 1997] [Li 1996] [Gys 1997] puisqu'il répond à l'objectif d'analyser des faits en fonction des dimensions.

Dans cette section, nous présentons les concepts de base de la modélisation multidimensionnelle, ainsi que une brève description pour la notion de « cube de données ». Enfin, nous focalisons sur les deux implémentations de modèle multidimensionnel : le schéma relationnel et le schéma multidimensionnel.

2. 7. 1 Concepts de base de la modélisation multidimensionnelle

Les concepts de base de la modélisation multidimensionnelle sont : fait, dimensions et mesures [Bal 1998] :

Fait

Un fait représente le sujet de l'analyse. Il se compose d'un ensemble de mesures et de contextes de données. Chaque fait est modélisé dans une table, nommée table de faits, qui contient des clés vers toutes les tables de dimensions modélisant les activités mesurées et un ensemble de mesures correspondantes à ces activités analysées.

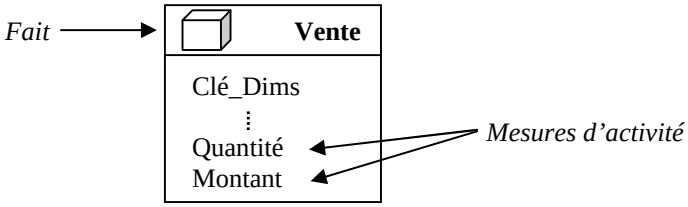


Figure 2. 14 Exemple de table de faits

Dimension

Une dimension est une collection de données qui décrivent une perspective de l'analyse. Les dimensions déterminent l'environnement contextuel pour les faits. Chaque dimension est modélisée dans une table, nommée table de dimension, qui contient un ensemble d'attributs, correspondants à des champs textuels descriptifs, et une clé primaire unique qui le reliée avec la ou les tables des faits à lesquelles cette dimension est attachée. Cette clé primaire correspond exactement à l'une des clés de ces tables des faits.

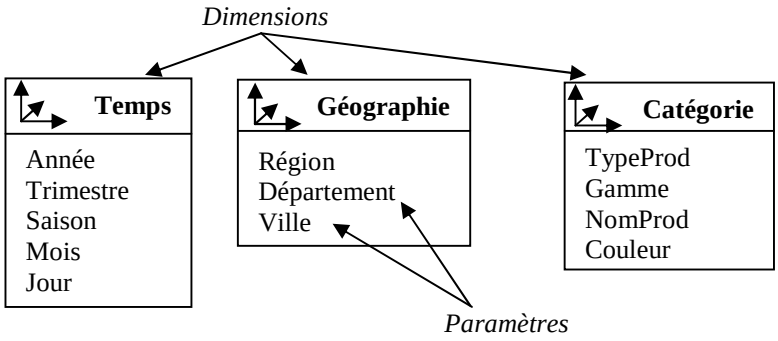


Figure 2. 15 Exemple de tables de dimensions

Les attributs d'une dimension sont organisés dans des niveaux hiérarchiques, afin de permettre à l'utilisateur d'examiner les mesures des faits à différents niveaux de détail, de « descendre » dans les données, en allant du niveau global au niveau plus fin. D'ailleurs, les attributs sont ordonnés par une relation « *est_plus_fin* », notée « $A_1 \rightarrow A_2$ ». Par exemple, la dimension Temps peut être organisée dans la hiérarchie suivante : Année → Semestre → Trimestre → Mois → Semaine → Jour, soit six niveaux hiérarchiques. Cependant, la granularité d'une dimension correspond au nombre de ses niveaux

hiérarchiques. Par exemple, la dimension Temps est organisée dans six niveaux hiérarchiques, donc elle est de granularité six.

Mesure

Une mesure est un attribut numérique d'un fait qui représente les différentes valeurs de l'activité analysée. Les mesures peuvent être des mesures numériques additives, semi additives (additionnée pour certaines dimensions) ou non additives.

2. 7. 2 Cube de données

Le cube de données offre une abstraction très proche de la façon dont l'analyste voit et interroge les données. Il est constitué d'un ensemble de cellules, où chaque cellule représente un fait. Ce dernier est décrit par des descripteurs catégoriels selon plusieurs axes d'analyse, appelés dimensions, et est observé par un ou plusieurs indicateurs, appelés mesures. Alors qu'une mesure est souvent une valeur additive, une dimension repose sur un ensemble fini de modalités qui représentent des descripteurs catégoriels. La figure 2. 16 montre un exemple de cube de données. Dans ce dernier, les axes d'analyse sont associés aux dimensions *Profession*, *Produit* et *Statut*. Chaque fait est représenté par une cellule décrite par un ensemble de modalité provenant des différentes dimensions. La cellule contient le *montant moyen des salaires* et *l'effectif des personnes*. Ces derniers indicateurs représentent les mesures du cube [Ben 2006a].

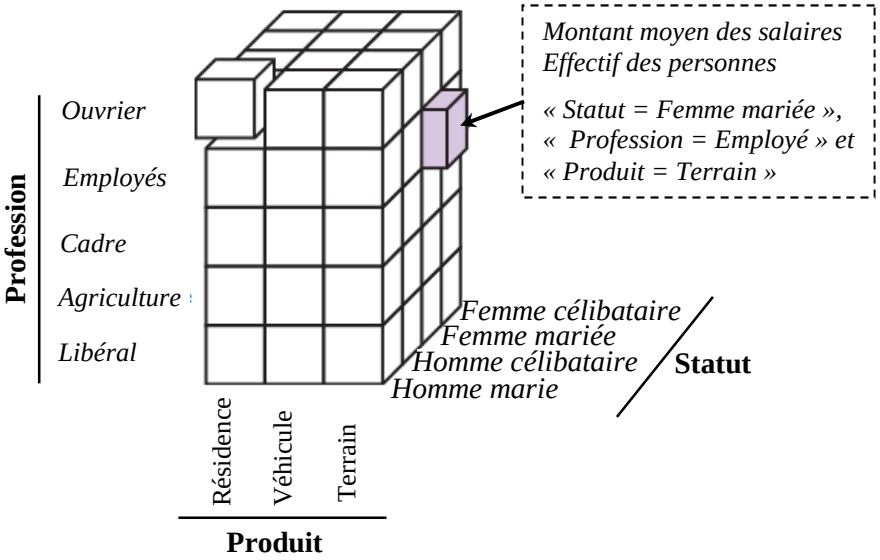


Figure 2. 16 Exemple de cube de données

D'une manière générale, une dimension comporte également plusieurs hiérarchies impliquant différents niveaux de précision dans la description des faits. De telles hiérarchies permettent d'observer des indicateurs selon plusieurs niveaux de granularité et de construire

des agrégats à partir des faits du niveau plus fin. Ainsi, un cube de données représente une structure multidimensionnelle comprenant une organisation hiérarchique des données. Cette structure est simple à manœuvrer et est capable de supporter des opérations d'analyses et d'exploration, en vue de répondre à des fins décisionnelles.

2. 7. 3 Implémentation de modèle multidimensionnel

Selon la façon dont le cube de données est stocké, le modèle multidimensionnel est implémenté physiquement selon deux façons [Cab 1998] [Moh 1999] : la première consiste à utiliser une base de données relationnelle (schéma relationnel) tandis que la deuxième consiste à utiliser une base de données multidimensionnelle spécialisée, ou encore un ensemble de tableaux multidimensionnelles (schéma dimensionnel). Dans la suite, nous présentons brièvement chaque approche.

2. 7. 3. 1 Schéma relationnel

Ce schéma stocke les données de l'entrepôt dans des tables relationnelles spécialisées, appelées tables de faits et tables de dimensions. L'organisation de ces tables peut être effectuée selon l'un des ces trois schémas suivants :

Schéma en étoile

Dans ce type de schéma, les mesures sont représentées par une table de faits et chaque dimension par une table de dimension. La table de faits référence les tables de dimensions en utilisant une clé étrangère pour chacune d'elle et stocke les valeurs des mesures pour chaque combinaison de clés. Autour de cette table de faits figurent les tables de dimensions qui regroupent les caractéristiques des dimensions. Les tables de dimensions sont généralement dénormalisées afin de minimiser le nombre de jointures nécessaires pour évaluer une requête.

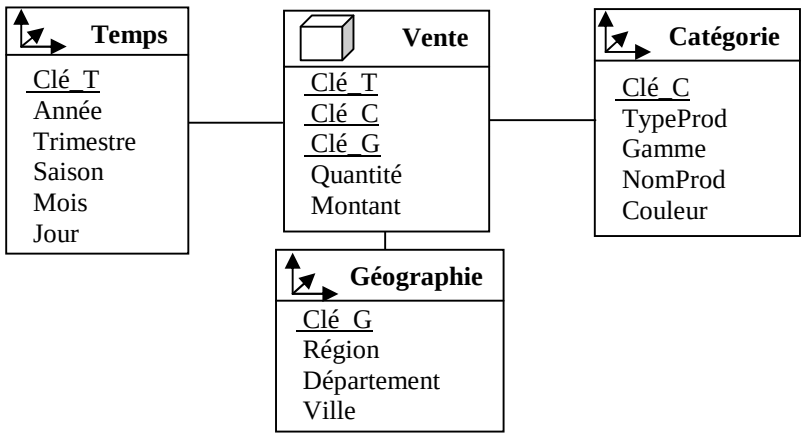


Figure 2. 17 Exemple d'un schéma en étoile modélisant les analyses des quantités et des montants selon trois dimensions : Temps, Catégorie et Géographie.

Schéma en flocon

Il correspond à un schéma en étoile dans lequel les dimensions ont été normalisées, faisant ainsi apparaître des hiérarchies des dimensions de façon explicite. La normalisation permet un gain d'espace de stockage en évitant la redondance de données, mais engendre une dégradation des performances, dans la mesure où elle multiple le nombre de jointure à effectuer pour l'analyse.

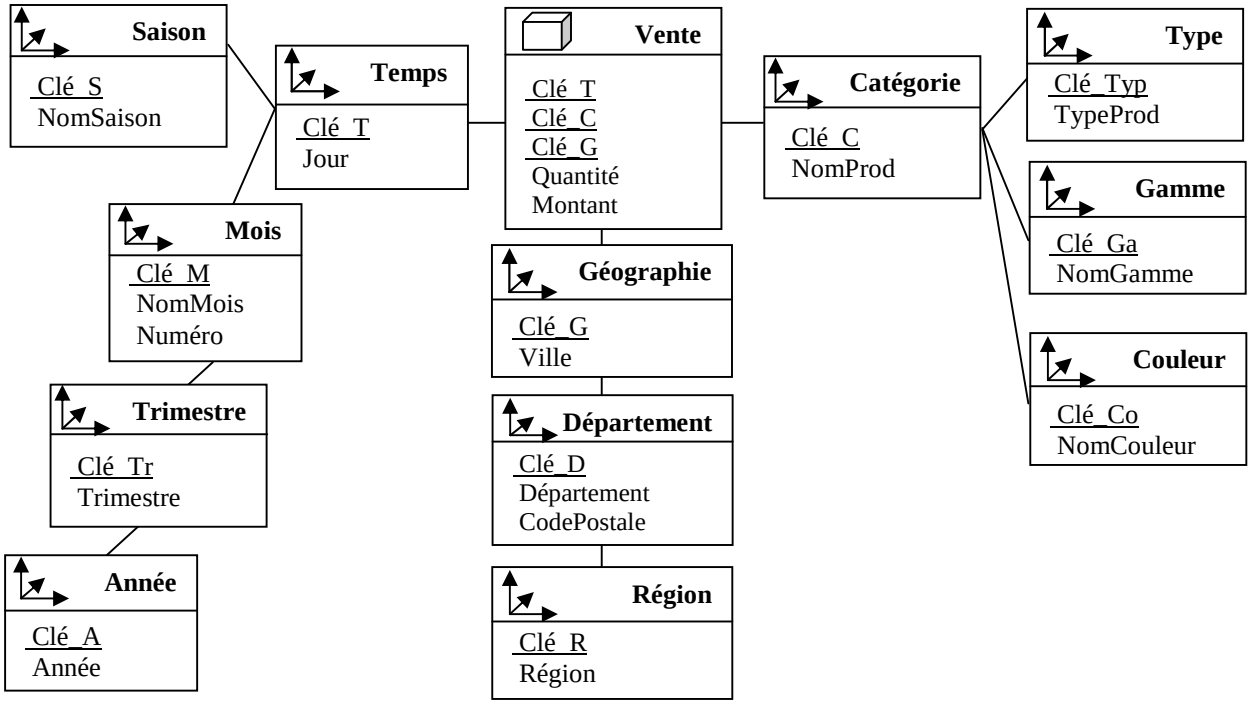


Figure 2. 18 Exemple d'un schéma en Flocon qui décrit le modèle en étoile de la figure précédente en dénormalisant chacune de ces dimensions, formant ainsi une sorte de flocon.

Schéma en constellation des faits

Le schéma en constellation des faits fait coexister plusieurs tables de faits qui partagent des dimensions communes (hiérarchisées ou non). Il s'agit de fusionner plusieurs schémas en étoile qui utilisent des dimensions communes.

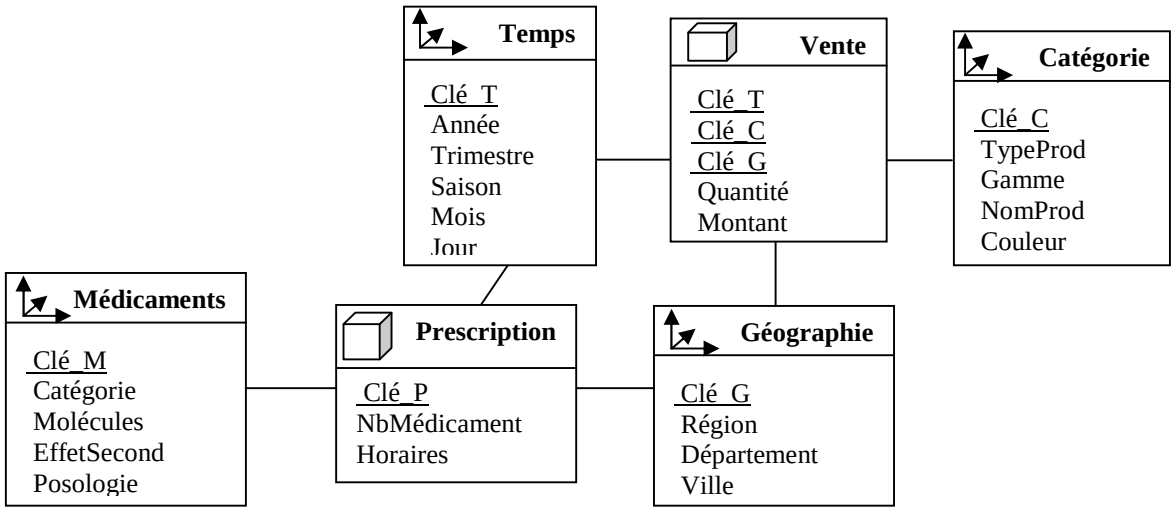


Figure 2. 19 Exemple d'un schéma en constellation qui décrit une constellation constituée de deux schémas en en étoile : l'un correspond aux ventes et l'autre analyse les prescriptions. Dans ce schéma la dimension Temps est partagée par deux tables de fait Vente et Prescription.

2. 7. 3. 2 Schéma multidimensionnel

Ce schéma stocke les données de l'entrepôt dans des structures multidimensionnelles natives, appelées tableaux multidimensionnels (ou *Multidimensional Array*). Dans la littérature, les termes de cube, hypercube et table multidimensionnelle sont utilisés de manière interchangeable. Le terme d'hypercube est généralement utilisé pour désigner des structures à deux, à trois ou à plus de trois dimensions. Donc, un hypercube de données est un Tableau à *n* dimensions. Chaque dimension possède une hiérarchie associée de niveaux de consolidation. Chaque position dans un tableau multidimensionnel, correspondant à une interaction de toutes les dimensions, est appelée une cellule. Ces dimensions peuvent être affinées, décomposées en hiérarchie, afin de permettre à l'utilisateur d'examiner les mesures à différents niveaux de détail. L'intérêt de cette approche est qu'il exige un espace de stockage de données très petit puisque les données sont groupées d'une manière compacte dans des tableaux multidimensionnels, ainsi que les temps d'accès sont optimisés, mais cette approche nécessite de redéfinir des opérations pour manipuler ces structure multidimensionnelles.

2. 8 La manipulation des données de l'entrepôt de données

L'entrepôt de données a pour objectif final l'analyse de données en vue de la prise de décision. Cette analyse peut être qualifiée d'exploratoire puisqu'il s'agit de détecter, de décrire et d'expliquer les points intéressants en navigant dans les données de l'entrepôt. Pour réaliser cette navigation, différentes opérations s'appliquent aux données multidimensionnelles stockées dans l'entrepôt [Mar 1998], [Tes 2000], [Han 2000],

[Agr 1995a]. Ces opérations permettre de visualiser ces données sous forme d'un cube de données qui se présente, à l'écran de l'utilisateur, sous différentes vue (opérations agissant sur la structure) et différents niveau de détail (opérations agissant sur la granularité).

2. 8. 1 Opérations agissant sur la structure

Les opérations agissant sur la structure visent à représenter une vue différente du cube en fonction de l'analyse à effectuer. Elles sont regroupées sous le nom de restructuration. Tout cube obtenu par une opération de restructuration d'un cube initial contient tout ce qu'il faut pour régénérer le cube initial par restructuration réciproque. Ces opérations sont : Pivot, Silce, Dice, Switch, Split, Nest, Push, et Pull, etc.

- *Pivot (Retate)* : Elle consiste à faire effectuer à un cube une rotation autour d'un des trois axes passant par le centre de deux faces opposées, de manière à présenter un ensemble de faces différentes. C'est en quelque sorte une sélection de faces et non de membres.
- *Slice & Dice* : L'opération *Slice* permet de sélectionner une seule dimension d'un cube de données. Ainsi, le résultat de cette opération est un sous cube (tranche de cube) contenant les données correspondantes à la dimension sélectionnée. Tandis que l'opération *Dice* permet de définir un sous cube de données en sélectionnant deux dimensions ou plus.
- *Switch* : Elle consiste à inter-changer la position des membres d'une dimension, de manière à permuter deux tranches du cube.
- *Split* : Elle consiste à présenter chaque tranche du cube, et à passer d'une représentation tridimensionnelle d'un cube à sa représentation sous la forme d'un ensemble de tables. D'une manière générale, cette opération permet de réduire le nombre de dimensions d'une représentation. On notera que le nombre de tables résultant d'une opération Split dépend des informations contenues dans le cube de départ et n'est pas connu à l'avance.
- *Nest* : Cette opération permet d'imbriquer des membres d'une dimension. L'un de ses intérêts est qu'elle permet de grouper sur une même représentation bi-dimensionnelle toutes les informations (mesures et membres) d'un cube, quel que soit le nombre de ces dimensions. L'opération réciproque, « *Unnest* », reconstitue une dimension séparée à partir des membres imbriqués.
- *Push* : Cette opération consiste à combiner les membres d'une dimension aux mesures du cube, et donc de faire passer des membres comme contenus de cellules. L'opération Push est nécessaire pour permettre le traitement uniforme des dimensions et mesures.
- *Pull* : C'est l'inverse de l'opération Push. Elle permet la création d'une nouvelle dimension pour un membre spécifique de chaque mesure. L'opération Pull est utile pour

convertir les mesures en dimensions pour que ces mesures puissent être utilisées dans la fusion et la jointure. Elle est aussi nécessaire pour le traitement symétrique des dimensions et mesures.

2. 8. 2 Opérations agissant sur la granularité

Les opérations agissant sur la granularité des données analysées, permettent de hiérarchiser la navigation entre les différents niveaux de détail d'une dimension, appelés niveaux de granularité. Les opérations permettant la hiérarchisation sont : Roll-up et Drill-down. Ces opérations autorisent l'analyse de données à différents niveaux d'agrégation en utilisant des hiérarchies associées à chaque dimension.

- *Roll-up (Drill-up ou Scale-up)* : Elle permet de représenter les données du cube à un niveau plus haut de granularité en respectant la hiérarchie de la dimension. Pour indiquer la façon de calculer les données du niveau supérieur à partir de celles de niveau inférieur, une fonction d'agrégation (moyenne, la somme, etc.) en paramètre de l'opération est utilisée.
- *Drill-down* : Cette opération est l'inverse de l'opération Roll-up. Elle consiste à représenter les données du cube à un niveau de granularité inférieur, donc sous une forme plus détaillée. Cette opération nécessite la connaissance des données au niveau inférieur.

2. 9 Le processus d'alimentation de l'entrepôt de données

L'alimentation est la procédure qui permet de transférer des données du système opérationnel vers l'entrepôt de données en les adaptant. Ainsi, avant de charger les données dans l'entrepôt, il est nécessaire de déterminer quelles données seront chargées, quelles transformations et vérifications seront nécessaires, la périodicité et le moment auxquels les transferts auront lieu. Puisque les entreprises possèdent une multitude de base de données de structures différentes, le processus d'alimentation de l'entrepôt qui englobe le traitement de cette diversité de sources de données devient de plus en plus complexe, coûteux, et nécessite beaucoup de temps pour sa mise en ouvre. En effet, ce processus est automatisé par des outils permettant d'assurer l'intégration et l'homogénéisation des données transférées dans l'entrepôt. Ces outils sont connus sous le nom ETL (*Extract-Transform-Load*), ou *Data Pumping*, et leurs fonctions principales peuvent être résumées en cinq étapes suivantes :

2. 9. 1 Extraction

L'extraction des données consiste à collecter les données utiles dans les systèmes opérationnels ; elle comprend la lecture et la compréhension des sources de données, ainsi que

la copie des données utiles pour les décideurs dans la zone de préparation. Pour optimiser cette phase certains outils commencent à fournir un type de fonctionnement qui permet de rafraîchir l'entrepôt avec les données modélisées ou ajoutées depuis la dernière extraction. Ce type de fonctionnement est appelé « *Changed data capture* ». Il consiste à réaliser des extractions différentielles en utilisant un mécanisme de marquage des données, souvent par examen de la date de dernière modification associée aux données. Il faut aussi planifier ces extraction à des moments opportuns afin d'éviter les saturations du système de production.

2. 9. 2 Fusion

Face à la diversité des sources de données de l'entreprise (fichiers, bases de données, etc.), il faut préparer chacune d'entre elles afin de pouvoir fusionner leurs informations. Un des grands problèmes de la fusion est de traiter les données venant de l'extérieur du système. Il faut maintenir une surveillance du système d'information pour pouvoir les identifier et s'assurer que ce sont les bonnes données qui sont recensées. De plus, la forme des données externes, qui est souvent totalement anarchique accentue la difficulté. Pour être utiles, ces données nécessitent un reformatage pour pouvoir les incorporer dans une forme exploitable pour l'entreprise.

L'intégrité des données est indispensable et nécessite la synchronisation des différents processus de fusion. Les problèmes lies à cette synchronisation peuvent être complexes, soit fonctionnellement, soit techniquement dans des environnements très hétérogènes. En conséquence, l'outil de fusion doit attaquer toutes sortes de sources de données sans être perturber et s'adapter aux futures modifications.

2. 9. 3 Nettoyage et transformation

Le nettoyage ou épuration de données a pour but de résoudre le problème de la consistance des données. Ces consistances peuvent être dues : (1) à la présence de données fausse dès leur saisie, (2) à la persistance de données obsolètes, (3) à la confrontation de données exactes, sémantiquement identiques, mais syntaxiquement différentes. Par conséquent, les outils de nettoyage de données de données permettent de supprimer les doublons, les données erronées, inconsistantes et bruyantes, les valeurs manquantes et toutes autres forme d'inconsistance dans l'ensemble de données collectées avant de les chargées dans l'entrepôt de données. Il existe deux classes d'outils de nettoyage de données : Les outils de *migration de données* permettent à des règles simples de transformation d'être spécifiées, et les outils de *d'épuration de données* utilisent une connaissance sur un domaine spécifique pour faire épurées les données.

Même nettoyées, les données doivent être formatées de manière à être acceptées par l'entrepôt de données. Pour cela, une étape de transformation est nécessaire. Il s'agit parfois de revoir la taille des champs, leur type, l'ordre de ces champs au sien de l'enregistrement.

2. 9. 4 Filtrage et chargement

Le chargement est la dernière phase de l'alimentation de l'entrepôt de données. C'est une phase délicate notamment lorsque les volumes sont importants. Pour obtenir de bonnes performances en chargement, il est impératif de maîtriser les structures du SGBD (les tables et surtout les index) associées aux données chargées afin d'optimiser au mieux ce processus. Les techniques de parallélisation optimisent les chargements lourds. Pour les mettre en œuvre, des utilitaires particuliers existent chez la majorité des éditeurs de bases de données.

Lors de la phase de filtrage, nous ne sélectionnons que les enregistrements qui correspondent à certains critères et qui seront transférés sur le système cible afin d'alimenter la base décisionnelle.

2. 9. 5 Post-traitement

Le post-traitement a lieu après le chargement. Cette phase consiste à compléter le chargement de l'entrepôt de données par la génération des données dérivées, ainsi que de calculer les données agrégées pour compléter les tables d'agrégats.

2. 10 Conclusion

Dans ce chapitre, nous avons évoqués les principaux concepts liés à la conception et l'exploitation des entrepôts de données. Ces derniers ont apportés une solution adéquate et efficace aux problèmes de stockage et de la gestion de données ; ce sont des bases centralisées de grands volumes de données, historisées, organisées par sujet et consolidées à partir de diverses sources d'informations [Inm 1996] [Kim 1996]. En plus de sa vocation de stockage, les données de l'entrepôt sont organisées et préparées pour supporter un processus d'analyse de données. Cette analyse est souvent qualifiée d'exploration puisqu'il s'agit de naviguer dans l'ensemble de données de l'entrepôt en vue d'extraire des nouvelles corrélations, ou plus précisément des nouvelles connaissances, auparavant inconnues et potentiellement utiles pour la prise de décisions stratégiques.

La tâche de découverte de connaissances à partir des grands volumes de données est effectuée par des techniques dites de Data Mining. Dans le chapitre suivant, nous effectuons un survole sur les principaux techniques d'extraction de connaissances ou de Data Mining existantes dans la littérature.

CHAPITRE 3

Techniques d'Extraction de Connaissances à partir de Données

3. 1 Introduction

Depuis l'explosion des capacités de stockage informatique au moins à partir du début des années 1990, la question de l'analyse de grands volumes de données s'est imposée. Un nouveau domaine de recherche, appelé « *Extraction de Connaissance à partir de Données* », qui tente de répondre à cette question en mettant au point des techniques de Data Mining capables d'analyser des grands volumes de données, ainsi émergé. En effet, l'Extraction de Connaissances à partir de Données (ECD) est un processus consistant à extraire des connaissances implicites, inconnues et potentiellement intéressantes à partir des grands volumes de données. Le processus de l'ECD se décompose en trois étapes principales : préparation de données, application des techniques de Data Mining et interprétation des résultats obtenus.

Ce chapitre, destiné à présenter quelques techniques de Data Mining, est subdivisé en trois sections. La première section (3. 2) est consacré à la présentation des techniques les plus répondus pour résoudre le problème de classification de données ; citant la méthode des *k*-Plus Proches Voisins, les arbres de décision, les réseaux bayésiens et les réseaux de neurones artificiels. Dans la deuxième section (3. 3), nous abordons le problème de Clustering ; en présentant une classification des différents algorithmes de Clustering, les mesures de similarité, dissimilarité et distance les plus fréquemment rencontré dans ces différents algorithmes, ainsi que quelques exemples d'algorithmes de Clustering (*k*-means, Classification Hiérarchique Ascendante et les cartes topologiques de Kohonen). Enfin, dans la troisième section (3. 4), nous expliquons le problème de recherche des règles d'association en détaillant ainsi le principe de son algorithme de base, l'algorithme Apriori.

3. 2 Classification

Parmi les techniques prédictives de Data Mining, on distingue la classification. C'est un processus supervisé à deux étapes. Pendant la première étape, un fragment de l'ensemble de

données, appelé ensemble d'apprentissage, est employé pour extraire des modèles précis qui place les données de l'ensemble d'apprentissage dans des classes (ou groupes) prédéfinies par les utilisateurs. Pendant la deuxième étape, le modèle est employé pour classifier n'importe quelle nouvelle donnée ou n'importe quel nouvel ensemble de données, ou pour extraire des règles de classification plus précises. En effet, pour une classification réussite, il suffit de prendre un ensemble de classes bien défini et un ensemble d'apprentissage qui contient des exemples déjà classés [Han 2000].

Les techniques de classification les plus populaires sont [Lar 2005] [Pre 2005] [An 2006] : les *k*-Plus Proches Voisins, les arbres de décision, les réseaux bayésiens et les réseaux de neurones artificiels. Dans cette section, nous détaillons chaque technique après avoir présenté une définition formelle pour le problème de classification.

3. 2. 1 Définition formelle

Soit D une base de données. Chaque individu $X_i \in D$ est d'écrit par P attributs a_j dans un espace de P-dimensions (j : 1, 2, …, P) : $X_i = [\ X_i^{\ 1}, X_i^{\ 2}, \ldots, X_i^{\ P}]$, tel que $X_i^{\ j}$ représente la valeur de l'attribut a_j pour l'individu X_i. On dispose d'un ensemble d'exemples $\mathcal{X} \subset D$ de n individus étiquetés par sa classe $Y_i \in \mathcal{Y}$, où $\mathcal{Y}$ est un ensemble fini de valeurs des classes ; il s'agit d'un attribut qualitatif , appelé attribut classe, pouvant prendre un nombre fini de valeurs. Un problème de classification consiste, en s'appuyant sur l'ensemble d'exemples $\mathcal{X} = \{ (X_i, Y_i)_{i \in \{1,\ldots,n\}} / X_i \in D \wedge Y_i \in \mathcal{Y} \}$, à prédire la classe de toute nouvelle donnée $x \in D$.

Ainsi, l'application de la classification permet de construire un modèle (ou classeur) à partir d'un ensemble d'exemples déjà classés et, ensuite, d'utiliser se modèle pour prédire la classe des nouvelles données.

3. 2. 2 Quelques méthodes de classification

3. 2. 2. 1 *k*-Plus Proches Voisins

La méthode des *k*-Plus Proches Voisins (*k-PPV)*, ou encore *k-Nearest Neighbor (k-NN),* est une méthode dédiée à la classification qui peut être étendue à des tâches d'estimation. La méthode des *k-PPV* est une méthode de raisonnement à partir de cas. Elle part de l'idée de prendre des décisions en recherchant en mémoire un ou des cas similaires déjà résolus. Elle est simple à comprendre sur le plan mathématique ; elle a fait ses preuves et est souvent utilisée en comparaison avec des méthodes plus récentes [Gri 2006].

Contrairement aux autres méthodes de classification qui seront étudiées dans cette section (arbres de décision, réseaux bayésiens et réseaux de neurones artificiels), il n'y a pas d'étape d'apprentissage consistant en la construction d'un modèle à partir d'un ensemble d'apprentissage. C'est l'ensemble d'apprentissage, associé à une fonction de distance et d'une fonction de choix de la classe en fonction des classes des voisins les plus proches, qui constitue le modèle. L'algorithme générique de la méthode de classification par *k-PPV* est le suivant :

Algorithme 3. 1 Classification par *k-PPV*

Paramètre : le nombre k de voisins ;

Donnée : un ensemble d'exemples $\mathcal{X}$ de n enregistrement classés dans des classes $Y_i \in \mathcal{Y}$, où $\mathcal{Y}$ est un ensemble fini de valeurs des classes ;

Entrée : un enregistrement x de classe inconnu

- Déterminer les k plus proches enregistrements de x, en calculant les distances entre x et tous les enregistrements de l'ensemble d'exemples $\mathcal{X}$ selon une fonction de distance ;
- Combiner les classes de ces k plus proches enregistrements de x en une classe c selon une fonction de combinaison ;

Sortie : la classe de x est $c(x) = c$;

Nous présentons, dans la suite, les différents choix possibles pour la définition d'une fonction de distance et pour le mode de sélection de la classe du cas présenté (la fonction de combinaison).

Le choix d'une fonction de distance

Le choix de la distance est primordial au bon fonctionnement de la méthode. Selon le type du champ de l'enregistrement et de l'attribut classe, plusieurs distances sont définies ; citant en exemple, la distance Euclidienne, la distance de Manhattan, la distance Minkowski, le coefficient de Jaccord, L'indice de Russel et Rao, etc. Dans la section 3. 3. 3, une présentation détaillée de ces distances est réalisée. Cependant, dans le cas de la méthode de classification par *k-PPV*, la distance Euclidienne est la plus commune [Lar 2005] :

$$d_{\text{Euclidienne}}(X_i, Y_j) = \sqrt{\sum_{k=1}^{P} (X_i^k - Y_j^k)^2}$$

Où $X_i = [X_i^1, X_i^2, \ldots, X_i^P]$ et $Y_j = [Y_j^1, Y_j^2, \ldots, Y_j^P]$ représentent les P valeurs d'attributs des deux enregistrements.

Le choix d'une fonction de combinaison

Une fois les distances entre x et les enregistrements de l'ensemble d'exemples $\mathcal{X}$ sont calculées et les k plus proches enregistrements de x sont définis, nous avons besoin d'établir comment combiner les classes de ces k plus proches enregistrements de x pour fournir une décision de la classification pour cet nouvel enregistrement, x. Pour cela nous avons besoin d'une fonction de combinaison. La méthode la plus simple est de chercher le cas le plus proche et de prendre la même décision. C'est la méthode *1-PPV* (*1-NN*) du plus proche voisin. Si cette méthode peut fournir de bons résultats sur des problèmes simples pour lesquels les enregistrements sont bien répartis en groupes denses d'enregistrements de même classe, en règle générale, il faut considérer un nombre de voisins plus important pour obtenir de bons résultats. En effet, le « *Vote Simple* » et le « *Vote Pondéré* » sont les fonctions de combinaisons les plus populaires pour sélectionner la classe d'un nouvel enregistrement [Lar 2005].

Le Vote Simple / Simple Unweighted Voting

- Avant de parcourir l'algorithme, on décide la valeur de *k*, c'est-à-dire combien d'enregistrements auront une voix dans la classification d'un nouvel enregistrement.
- Puis, on compare le nouvel enregistrement aux *k* voisins les plus proches, c'est à dire, aux *k* enregistrements qui sont à une distance minimum du nouvel enregistrement (pour la distance retenue), qui reçoivent chacun une voix.
- La classe du nouvel enregistrement est alors la classe ayant le maximum de voix.

Le Vote Pondéré / Weighted Voting

Est-il judicieux qu'un enregistrement X_i ait la même voix qu'un enregistrement Y_j plus proche au nouvel enregistrement? Peut-être pas. L'analyste peut choisir d'appliquer un Vote Pondéré, où les voisins les plus proches ont une plus grande voix dans la décision de la classification que les voisins plus distants.

3. 2. 2. 2 Arbres de décision

Puisque ils sont très performants et elles génèrent des procédures de classification exprimables sous forme de règles, les arbres de décision constituent l'une des techniques les plus intuitives et populaires de Data Mining. Il s'agit de trouver un partitionnement des

individus que l'on présente sous la forme d'un arbre. Donc, l'objectif est de subdiviser progressivement l'ensemble d'apprentissage en plusieurs sous-ensembles de plus en plus homogènes. Ce partitionnement est effectué selon le choix de l'attribut qui discrimine le mieux un groupe d'individus et d'un test sur cet attribut. Chaque chemin de l'arbre, partant de la racine à une feuille, est traduit par une règle de type « IF *condition* THEN *résultat* », construisant ainsi un « système de règles de décision » qui permettra de prédire la classe des nouveaux individus.

Dans cette section, nous présentons une brève définition pour un arbre de décision. Puis, nous détaillons le processus de construction de cet arbre ; en mettant en évidence les particularités des différents algorithmes d'arbres de décision les plus connus et les plus utilisés en Data Mining. Parmi ces algorithmes, on trouve l'algorithme ID3 (*Inductive Decision Tree*) [Qui 1986], C5.0 (version plus récente de ID3 et C4.5 [Qui 1993]), CART (*Classification And Regression Tree*) [Bre 1984] et la méthode CHAID (*CHi-square Automation Interaction Detection*) [Har 1975].

Définition d'un arbre de décision

Un arbre de décision, ou « *Decision Tree* » en anglais, est une représentation graphique d'une procédure de classification exprimable sous forme d'une série de règles. Chaque nœud interne correspond à un test sur un attribut, chaque branche partant d'un nœud correspond à une valeur de ce test et chaque feuille représente une classe. Le nœud le plus haut dans l'arbre est le nœud racine, il correspond à tout l'ensemble d'apprentissage. La figure ci-dessous montre un exemple d'un arbre de décision.

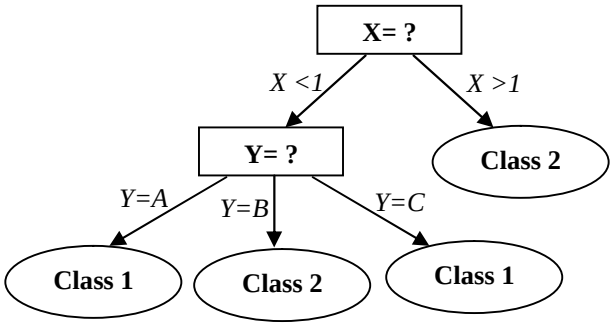


Figure 3. 1 Exemple d'un arbre de décision

Le nombre de branches descendantes à partir de chaque nœud dépend des résultats du test effectué à ce niveau. Selon ce nombre, Il existe deux types d'arbre de décision : Les arbres binaires (dans le cas de deux branches descendantes à partir de chaque nœud) et les arbres n-aires (dans le cas de plusieurs branches).

Construction d'un arbre de décision

Pour construire un arbre de décision, l'idée de base est de diviser, de façon récursive, l'ensemble d'apprentissage en plusieurs groupes d'individus de plus en plus homogènes. Débutant par un nœud racine contenant tout l'ensemble d'apprentissage, la construction de l'arbre est effectuée selon les étapes suivantes :

1. Sélectionner l'attribut qui discrimine le mieux un ensemble de données, ou un nœud, en deux ou plusieurs sous-ensembles. Cette sélection est basée sur des « *Critères de sélection d'attributs* ».
2. Partitionner l'ensemble de données entre les nœuds fils selon la satisfaction des « *tests de séparation* » sur l'attribut sélectionné.
3. Définir une « *condition d'arrêt* » permettant de décider qu'un nœud est terminal (feuille).
4. Affecter chaque feuille à une classe majoritaire.
5. « *Élaguer l'arbre de décision* » obtenu.

Nous allons préciser ces différentes étapes en mettant en évidence les particularités de différents algorithmes.

Les critères de sélection d'attribut

Il existe plusieurs critères qui permettant de sélectionner l'attribut le plus discriminant d'un ensemble de données. Parmi ces critères nous citons ci-dessous les plus utilisés :

L'entropie ou le gain d'information : Proposé par [Sha 1948] et utilisé par l'algorithme ID3 et ces variantes C4.5 et C5.0. La notion d'entropie permet de mesurer l'hétérogénéité d'un ensemble de données. Plus le nœud est pur, plus l'entropie est basse. Il permet, également, de manipuler tous types d'attributs.

L'entropie d'un nœud S est calculée comme suit :

$$\text{Entropie}(S) = -\sum_{i=1}^{k} f_i \log_2(f_i)$$

Où les f_i représentent les fréquences relatives dans le nœud S des k classes à prédire.

Soient les ensembles $\{S_1, S_2, ..., S_v\}$ formant une partition pour le nœud S, par un attribut A. L'information portée par cette partition est alors la moyenne pondérée des entropies :

$$E(A) = \sum_{i=1}^{v} \frac{n_i}{n} \text{Entropie}(S_i)$$

Où n est l'effectif de nœud S et n_i l'effectif de chaque nœud S_i (i : 1,…, v).

Finalement, le gain d'informationnel associé à l'attribut A est :

$$\text{Gain}(S, A) = \text{Entropie}(S) - E(A)$$

Plus ce gain est grand, plus l'attribut est pertinent. Donc, on cherche à maximiser le gain d'information.

Le critère du Chi-2 : Exploité par l'algorithme CHAID, ce critère de sélection est utilisé dans le cas des attributs qualitatifs. Il permet de vérifier la conformité d'un phénomène aléatoire à une loi de probabilité posée comme hypothèse, il s'appuie sur la comparaison entre les fréquences observées pour chacune des classes et les fréquences théoriques. Le choix de l'attribut qui partitionne un nœud en plusieurs nœuds fils est fait en sorte de maximiser la valeur de Chi-2.

L'indice de Gini : Proposé par [Bre 1984] en 1984 et utilisé par l'algorithme CART. L'indice de Gini mesure la probabilité que deux individus choisis de façon aléatoire dans un nœud, appartiennent à deux classes différentes. De ce fait, plus l'indice de Gini du nœud est bas, plus il est pur, et vice versa. On peut calculer l'indice de Gini comme suit :

$$\text{Indice de Gini} = 1 - \sum_{i=1}^{k} f_i^2$$

Où les f_i représentent les fréquences relatives dans le nœud S des k classes à prédire.

Chaque séparation en k nœuds fils $\{S_1, S_2, ..., S_k\}$ doit provoquer la plus grande hausse de pureté, donc la plus grande baisse de l'indice de Gini [Tuf 2003]. Autrement dit, il faut minimiser la quantité suivante :

$$\text{Indice de Gini (séparation)} = \sum_{i=1}^{k} \frac{\text{n}_\text{i}}{\text{n}} \text{Indice de Gini}(S_i)$$

Où n est l'effectif de nœud S et n_i l'effectif de chaque nœud S_i (i : 1,…, k).

Les tests de séparation

Les tests effectués à chaque nœud de l'arbre mettent en jeu plusieurs opérations, telles que les opérations de comparaison (=, $\neq$, <, >, $\leq$, $\geq$), les opérations logique ($\wedge$, $\vee$, $\neg$) et les opérations sur les ensembles ($\in$, $\notin$). Ces opérations sont utilisées selon le type

de l'attribut traité à chaque nœud de l'arbre. En effet, selon ce type, il existe plusieurs jeux de test :

Attributs qualitatifs : Dans le cas d'un attribut qualitatif, le nombre de nœuds fils crées est égale au nombre de valeurs prises par cet attribut (c'est-à-dire, le nombre de branches est égale au nombre de modalité). Ainsi, chaque branche de l'arbre est étiquetée par un test de la forme « *Attribut = Valeur* ».

Attributs continus ou discrets : Dans ce cas, il y a une infinité de tests envisageable. La première méthode consiste à découper l'ensemble des valeurs possibles pour l'attribut en n segments (ou tranches). Ensuite, essai les tests de la forme « *Attribut* $\in$ *Tranche* », « *Attribut* $\notin$ *Tranche* ». Tandis que, la deuxième méthode consiste à créer un seuil c et puis essai les tests de la forme « *Attributs* $\leq c$ », « *Attributs* $\geq c$ », etc.

Noter que les opérations logiques sont utilisées pour combiner plusieurs tests portés soit sur le même attribut soit sur plusieurs attributs.

Les conditions d'arrêt

Le nœud courant est terminal si et seulement si (liste non exhaustive) :

- Il n'y a plus d'attributs disponibles, c'est-à-dire que sur le chemin menant de la racine au nœud courant tous les tests disponibles ont été utilisés.
- Tous les exemples de l'ensemble de données courant sont dans une même classe.
- La proportion d'observations d'une classe est supérieure à un seuil prédéfini.
- Il n'existe pas de test ayant au moins k éléments sur chaque branche de l'arbre. Cette condition est utilisée par l'algorithme C5.0.
- Les deux premières conditions sont indiscutables, les autres conditions sont spécifiques aux différents algorithmes.

L'élagage de l'arbre de décision

Il est possible de poursuivre la croissance de l'arbre jusqu'à obtention d'un arbre complet A_{max} d'erreur réelle mesurée sur l'ensemble d'apprentissage la plus petite possible. Cependant, on risque d'être confronté à un problème de sur-apprentissage (sur spécialisation) : on a appris « par cœur » l'ensemble de données sur lequel l'apprentissage a été fait, mais on n'est pas capable de généraliser (prédire de nouvelles données).

Pour éviter le problème de sur-apprentissage, on applique une *procédure d'élagage*. Celle-ci est basée sur la suppression successive des branches les moins informatives en terme de discrimination entre les classes [Leb 1998]. Ainsi, on obtient un arbre plus petit avec un meilleur pouvoir de généralisation (même si l'erreur augmente sur l'ensemble d'apprentissage).

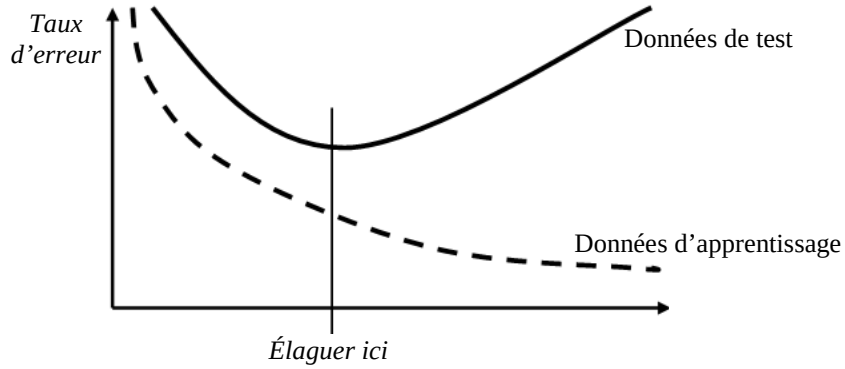


Figure 3. 2 Taux d'erreur d'un arbre en fonction de sa profondeur

L'élagage d'un arbre peut être appliqué de deux manières :

Le pré-élagage : Elle consiste à cesser de diviser un nœud quand celui-ci est jugé suffisamment pur. Autrement dit, on crée une feuille s'il existe une classe *c* suffisamment majoritaire sous ce nœud (par rapport à un seuil a priori). Cette méthode présente certains inconvénients, dont le principal est le risque de manquer de développer un arbre qui serait excellent (puisqu'elle ne prend en compte qu'un critère local à la feuille examinée) [Cor 2003]. C'est pourquoi on préfère souvent les méthodes d'élagage a posteriori.

Le post-élagage : Cette méthode a été introduite par [Bre 1984] en 1984 et elle est utilisée, notamment, par l'algorithme CART. Le principe de cette méthode consiste à construire, à partir de l'ensemble d'apprentissage, un arbre complet $A_{\max}$ de sorte que chaque feuille contienne peu d'individus. Puis, construire une séquence optimale de sous-arbres par élagage successif. Cette séquence se note $S = \{A_{\max},...,A_k,...,A_1\}$ où A_k est le sous arbre ayant k feuilles. Chaque sous-arbre A_k est optimal au sens suivant : son erreur mesurée sur l'ensemble d'apprentissage est minimal parmi les sous-arbres ayant le même nombre de feuilles. Parmi tout les sous-arbres de l'ensemble S, sélectionner le meilleur sous-arbre A^* au sens d'un critère. L'arbre CART utilise le critère suivant : le meilleur sous-arbre est celui qui présente la plus petite erreur mesurée sur l'ensemble de test.

3. 2. 2. 3 Réseaux bayésiens

Un réseau bayésien est un modèle probabiliste graphique proposé initialement par [Pea 1988]. Le succès de ces modèles est fortement lié à leur capacité de modéliser des systèmes complexes ou le raisonnement lorsque les situations sont incertaines. Graphiquement, un réseau bayésien est représenté par un graphe orienté sans circuit, dont à chaque nœud est associée une table de probabilités conditionnelles sur laquelle le modèle repose pour prédire des situations inconnues. Ces probabilités conditionnelles, constitues une connaissance a priori pour un modèle bayésien, sont estimées en se basant sur le théorème de bayes. Également, ces modèles manipulent une connaissance a priori concernant la structure du réseau. Cette structure peut être fournie par un expert ou déterminée par apprentissage à partir d'une base de données. Une variante simple des réseaux bayésiens est appelée réseaux bayésiens naïfs. Ces réseaux ont une structure simple et unique, avec la forte hypothèse naïve d'indépendance entre les variables.

Nous débutons cette section par une définition d'un réseau bayésien. Puis, nous présentons en détaille l'algorithme de classification à base des réseaux bayésiens naïfs, puisque ces derniers sont les plus appropriés pour le traitement des problèmes de classification [Fri 1996].

Définition

Un réseau bayésien, ou « *Bayesian network* » en anglais, est un graphe acyclique orienté, dont chaque nœud représente une variable (ou un attribut) et chaque arc représente une relation de dépendance entre les variables. A chaque nœud du réseau est associée une table de probabilités conditionnelles $[P(X^i / X^{P_a(X^i)})]_{i\in\{1,\ldots,P\}}$, où $P(X^i / X^{P_a(X^i)})$ est la probabilité conditionnelle du nœud i connaissant l'état de ces parents immédiat $P_a(X^i)$ dans le graphe. La figure ci-dessus montre un exemple d'un réseau bayésien.

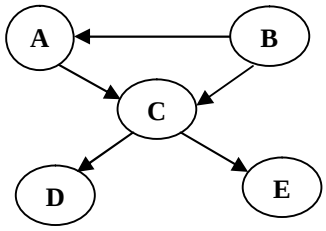


Figure 3. 3 Exemple d'un réseau bayésien

Un réseau bayésien naïf est la plus simple structure pour un réseau bayésien, il se compose de deux niveaux seulement. Le premier niveau contient un seul nœud parent et le second plusieurs enfants de ce nœud avec la forte hypothèse naïve d'indépendance entre les

nœuds enfants dans le contexte de leurs parents [Ben 2006b]. Pour le traitement des problèmes de classification, on considère le nœud parent comme une variable non observée précisant à quelle classe appartient chaque objet et les nœuds enfants comme étant des variables observées correspondants aux différents attributs spécifiant cet objet. La figure ci-dessus montre un exemple d'un réseau bayésien naïf.

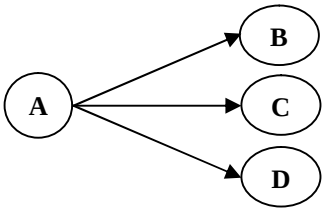


Figure 3. 4 Exemple d'un réseau bayésien naïf

Classification bayésienne naïve

Le principe de fonctionnement de la classification bayésienne naïve peut être résumé comme suit [Han 2000] :

Soit $\mathcal{X}$ une base d'exemples contenant n individus X_i décrits par P attributs : $X_i = [X_i^1, X_i^2, \ldots, X_i^P]$ (voir section 3. 2. 1). Soit C = $\{C_1, C_2, \ldots, C_m\}$ un ensemble de m classes. Le modèle permet de prédire la classe de toutes nouvelles données $X \in D$ comme étant la classe qui possède la plus grande probabilité à posteriori, conditionnée sur X. En d'autre terme, X est assigné à une classe C_i si est seulement si :

$$P(C_i / X) > P(C_j / X) \quad \forall \ 1 < i, j < m, i \neq j$$

Donc, on cherche à maximiser $P(C_i / X)$. Par le théorème de bayes, nous avons :

$$P(C_i / X) = \frac{P(X / C_i)P(C_i)}{P(X)}$$

Où $P(C_i / X)$ est la probabilité que X soit de classe C_i. $P(X / C_i)$ la probabilité qu'un élément de classe C_i soit X. $P(C_i)$ la probabilité d'observer la classe C_i. $P(X)$ la probabilité d'observer l'élément X.

Comme $P(X)$ est constante pour toutes les classes. Alors le problème revient à maximiser la quantité $P(X / C_i)P(C_i)$. Lorsque les probabilités à priori des classes sont inconnues, on suppose souvent quelles sont équiprobables (c'est-à-dire, $P(C_1) = P(C_2) = \ldots = P(C_m)$). Cependant, les probabilités $P(C_i)$ peuvent être estimées en comptant le nombre

d'occurrences de la classe C_i dans l'ensemble d'apprentissage. Donc, le problème est ramené à chercher une estimation pour la probabilité $P(X/C_i)$. Pour réduire le coût de calcul de cette probabilité, le modèle bayésien utilise une hypothèse naïve qui peut être résumé comme suit : « les attributs de X sont conditionnellement indépendants entre eux en connaissant la classe de X ». Par conséquent, on aura :

$$P(C_i / X) = \prod_{k=1}^{P} P(X^k / C_i)$$

Ainsi, pour estimer la probabilité $P(X/C_i)$ elle suffit d'estimer les probabilités $P(X^k/C_i)$ à partir de l'ensemble d'apprentissage.

Selon le type de l'attribut a_k, la probabilité $P(X^k/C_i)$ est estimée comme suit :

- Les probabilités conditionnelles pour les attributs discrets sont calculées à partir des fréquences en comptant le nombre d'apparitions de chaque valeur d'attribut avec chacune des valeurs que le nœud parent peut prendre ;
- Les attributs continus sont généralement traités en supposant qu'ils suivent une distribution de probabilité gaussienne. Donc, pour chaque valeur de classe C_i et chaque attribut continu a_k, nous devons calculer la moyenne μ et l'écart type σ qui vont nous servir pour le calcul de la fonction de densité de probabilité pour chaque valeur X^k de a_k comme suit :

$$f(X^k) = \frac{1}{\sqrt{2\pi}\cdot\sigma}\cdot e^{-\frac{(X^k-\mu)^2}{2\sigma^2}}$$

La classification bayésienne naïve est une méthode simple à mettre en œuvre. Bien qu'elle soit basée sur une hypothèse fausse en général (les attributs sont rarement indépendants), elle donne cependant des bons résultats dans les problèmes réels.

3. 2. 2. 4 Réseaux de neurones artificiels

Les réseaux de neurones artificiels sont des outils très utilisés pour la classification, la prédiction et la segmentation. Ils désignent un ensemble d'unités de calcul élémentaires (les neurones) organisées selon une architecture, dont le principe de fonctionnement est inspiré de celui du système nerveux humain. Une grande variété de réseaux de neurones a été conçue pour diverses applications en Intelligence Artificielle et en traitement de l'information. Nous nous limitons dans cette section à un type particulier de réseaux de neurones très utilisé en classification et en prédiction qui sont les Perceptrons MultiCouches (PMC).

Avant de présenter les PMC, nous commençons cette section par fournir une définition aux réseaux de neurones artificiels ainsi que à son unité de base, le neurone formel.

Définition

Un Réseau de Neurones Artificiel (RNA), ou encore « *Artificial Neural Network (ANN)* » en anglais, est, comme son nom indique, un ensemble de processeurs élémentaires, appelés neurones formels, groupés en réseau, de sorte que les signaux sortants des neurones deviennent des signaux entrants dans d'autres neurones. A chaque connexion entre deux neurones formels est associé un poids W_{ij} qui pondère le signal.

D'une manière générale, les réseaux de neurones artificiels se comportent, d'un point de vue de l'extérieur, comme une fonction qui traite des données (entrées) et produit une réponse correspondante (figure 3. 5).

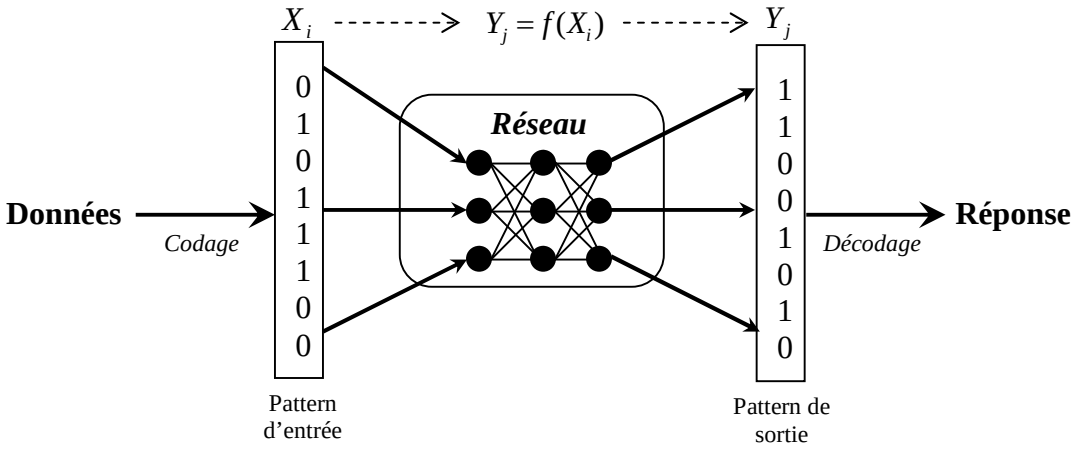


Figure 3. 5 Architecture fonctionnelle d'un réseau de neurones artificiels

Il a été présenté par MacCullogh et Pitts [Mac 1943], on s'inspirant de leurs travaux sur les neurones biologiques, le neurone formel constitue l'unité de base des réseaux de neurones artificiels. Son principe de fonctionnement peut être résumée en deux activités ; la première consiste à faire une somme pondérée des potentiels d'actions qui lui parviennent, puis s'active suivant une valeur de cette sommation pondérée. Si cette somme dépasse un certain seuil, le neurone est activé et transmet une réponse dont la valeur est celle de son activation. Si le neurone n'est pas activé, il ne transmet rien [Dav 1993].

D'une façon plus générale, on peut définir un neurone formel par les six éléments suivants (figure 3. 6) :

- *Les entrées* : Le neurone formel possède n+1 entrées, notée $(x_i)_{i=0,\ldots,n}$, qui peuvent être binaires ou réelles.

- *Les paramètres de pondération ou poids synaptiques* : Chaque entrée du neurone formel est pondérée par un poids, noté $(W_i)_{i=0,...,n}$, à valeur dans $\Re$.

- *Le biais d'entrée* : L'entrée x_0 est particulière : on la nomme le « Biais » ou « Seuil » et vaut toujours 1.

- *La fonction d'entrée totale ou de combinaison* : La fonction de combinaison, notée E, définit le pré-traitement effectué sur les entrées. En général, celle-ci peut être une somme pondérée des entrées du neurone :

$$E = h(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} w_i x_i$$

- *La fonction d'activation ou de transfère* : La fonction de transfert, notée $A = f(E)$, calcule l'état interne du neurone en fonction des son entrées totale. Cette fonction peut être :

  Une fonction binaire à seuil (la fonction de Heaviside) :

$$f(x) = \begin{cases} 1 & Si\ x \geq 0 \\ 0 & Si\ x < 0 \end{cases} \qquad \text{Où} \qquad f(x) = \begin{cases} 1 & Si\ x \geq 0 \\ -1 & Si\ x < 0 \end{cases}$$

  Une fonction sigmoïde :

$$f(x) = \frac{1}{1 + e^{-kx}} \qquad \text{avec k>0}$$

- *La fonction de sortie* : La fonction de sortie, notée $S = g(A)$, calcule la sortie du neurone en fonction de son état d'activation. En générale, cette fonction est considérée comme la fonction identité. On a alors $S = g(A) = A = f(E)$.

- *La sortie* : La sortie S du neurone qui peut être booléenne ou réelle.

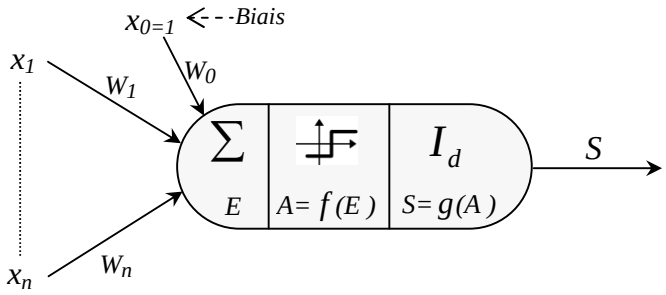


Figure 3. 6 Représentation générale d'un neurone formel

Perceptron

Introduit par Rosenblatt [Ros 1958] en 1958, le perceptron est le réseau le plus simple pour classer les individus d'un ensemble de données en deux classes. Comme il montre la figure 3. 7, le perceptron est constitué d'une couche de neurones d'entrées associés aux attributs de l'individu à classer (*variables explicatives*) et d'un neurone de sortie associé à l'attribut classe de l'individu à classer (*variable à expliquer*). Les neurones de la couche d'entrée ont une seule entrée et une seule sortie, égale à l'entrée. Le neurone de la couche de sortie a en entrée tous les nœuds de la couche d'entrée, avec une fonction de combinaison qui correspond à la somme pondérée de ces entrées et la fonction de transfert qui est en générale une fonction binaire à seuil. Les connexions entre les neurones d'entrées et le neurone de sortie sont pondérées par les poids W_i à valeur dans $\Re$. L'entrée x_0 (toujours vaut 1) est particulière, elle est pondérée par W_0 qui correspond au seuil de la fonction de transfert. La sortie S du perceptron est binaire, (-1 ou 1), expriment l'appartenance ou non à une classe.

$$S = \begin{cases} 1 & si \quad \sum_{i=1}^{n} W_i x_i > W_0 \\ -1 & \text{sinon} \end{cases}$$

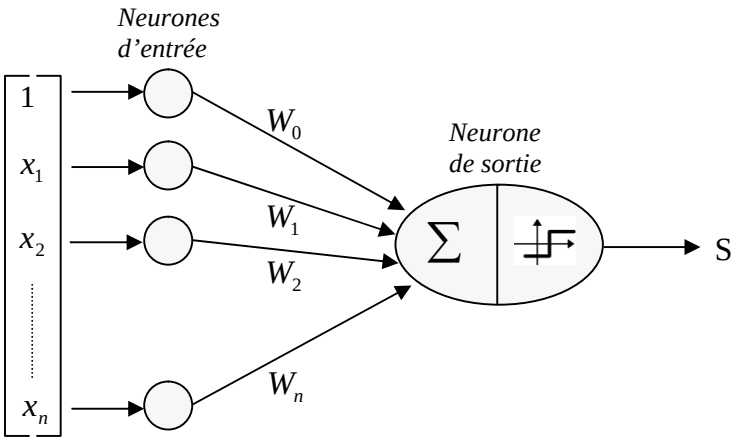


Figure 3. 7 Perceptron

Le Perceptron réalise, donc, une partition de l'espace de ses entrées en deux classes selon la valeur de sa sortie, la première pour laquelle il répond +1, et l'autre -1. La séparation de ces deux zones est effectuée par un hyperplan ou un plan dans un espace N-aires (figure 3. 8). On dira que ce perceptron effectue une *séparation linéaire* de son espace d'entrée. L'équation de la droite séparatrice est :

$$\sum_{i=1}^{2} W_i x_i = W_0$$

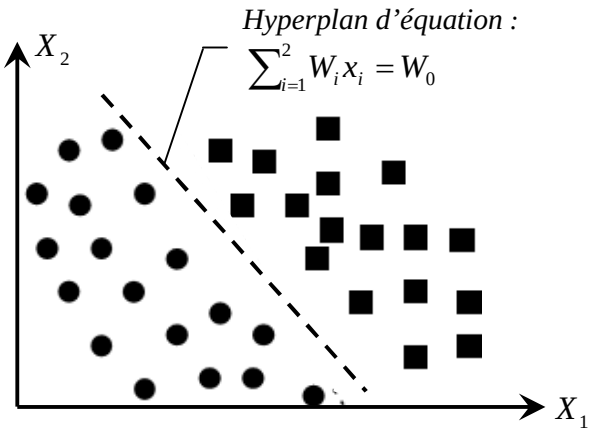


Figure 3. 8 Partitionnement réalisé par un Perceptron pour deux attributs

L'apprentissage est une phase du développement d'un réseau de neurones durant laquelle les paramètres de pondération du réseau sont estimés, afin que celui-ci remplisse au mieux la tâche qui lui est affectée [Dre 2002].

L'apprentissage du réseau est effectué plusieurs fois selon des règles d'apprentissage. Dans le cas de Perceptron, la modification des poids s'effectue selon la règle de Windrow-Hoff :

$$W_i(t+1) = W_i(t) + \alpha(d - S)x_i$$

Où α est le pas d'apprentissage, $X = [x_1, x_2, ..., x_n]$ est le vecteur des caractéristiques présenté à l'entrée du perceptron, d est la sortie désirée et S est la sortie calculée par le perceptron.

Algorithme 3. 2 apprentissage du perceptron

// t est le nombre d'itérations et $t_{\max}$ est le nombre maximum d'itérations.

Initialisation des poids $W = [W_0, W_1, ..., W_n]$ du réseau : elle est souvent faite aléatoirement ;

$t = 0$;

Tant que ($t < t_{\max}$) **faire**

Tirer au hasard un exemple (X, d) dans l'échantillon d'apprentissage ;

Calculer la sortie S du perceptron correspondant à l'entrée X ;

Pour i de 1 à n

$\Delta W_i = \alpha(d - S)x_i$;

$W_i = W_i + \Delta W_i$;

Fin Pour

$t = t + 1$ **;**

Fin Tant que.

Le perceptron présente certaines limites, qui ont été démontré par Papert et Minsky [Min 1969]. Par exemple, son incapacité à apprendre des classifications non linéairement séparables. En effet, pour partitionner l'ensemble d'individus en plusieurs classes non linéairement séparable, il faut grouper plusieurs Perceptrons en plusieurs couches. Cette organisation en couche fait apparaître un modèle très reconnu en classification, qu'est le Perceptron MultiCouche (PMC).

Perceptron MultiCouche

L'archétype des réseaux de neurones artificiels est le Perceptron MultiCouche (PMC). Il est particulièrement bien adapté à la découverte de modèles complexes et non linéaires. Comme son nom indique, le PMC est organisé en niveaux (ou couches). Il n'y a pas de connexions entre neurones d'une même couche. Les connexions ne se font qu'avec les neurones des couches en avales. Ceci nous permet d'introduire la notion de sens de parcours de l'information (ou de l'activation) au sien d'un réseau et donc définir les concepts de neurones d'entrée, neurones de sortie (figure 3. 9). On appelle *couche d'entrée* l'ensemble des neurones d'entrée qui est la seule dont les neurones ont leur état directement influencé par l'environnement. La *couche de sortie* est constituée de l'ensemble des neurones de sortie et c'est la seule qui fournit directement une réponse à l'environnement. Les couches intermédiaires n'ayant aucun contact avec l'extérieur sont appelées *couches cachées*.

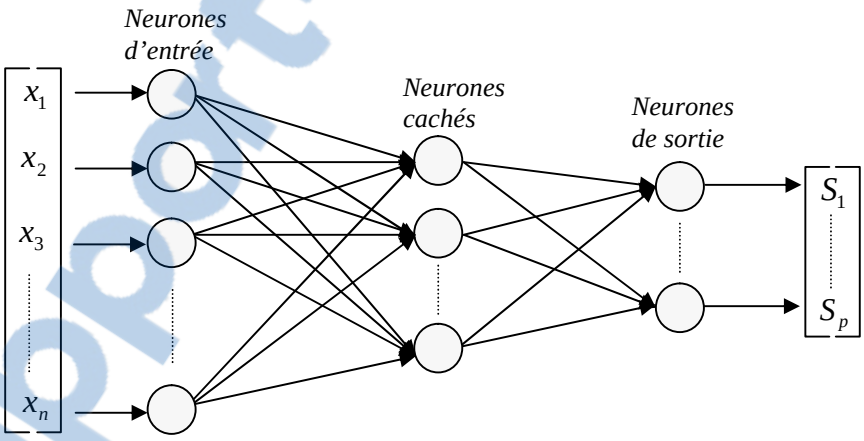


Figure 3. 9 Perceptron MultiCouche (PMC)

Chaque neurone de la couche d'entrée correspond à un attribut de l'individu à classer, chaque neurone de la couche de sortie correspond à une classe et les neurones des couches cachées n'ont pas de sémantique propre mais permettent de modéliser les problèmes non linéairement séparables. Le calcul de l'activité dans le réseau s'effectue en propageant l'exemple présenté en entrée de la couche d'entrée jusqu'à la couche de sortie. L'erreur, quant à elle, est calculée dans le sens inverse [Jod 1992]. En effet, l'exemple présenté en entrée est propagé de couche en couche jusqu'à la sortie, chaque neurone détermine son état grâce à sa

dynamique propre. L'état d'un neurone de sortie est interprété comme un degré d'appartenance (probabilité) de l'exemple présenté en entrée à la classe que code la sortie. La règle d'étiquetage utilisée dans un PMC consiste à attribuer à un exemple la classe identifiée par la sortie calculée maximale.

L'apprentissage des valeurs des poids se fait généralement à l'aide du principe de *rétropropagation du gradient de l'erreur* entre les sorties obtenues $S = [s_1, s_2, ..., s_p]$ et les sorties désirées $D = [d_1, d_2, ..., d_p]$ [Rum 1986]. Ce principe peut être résumé comme suit :

1. Initialisation aléatoire de la matrice des poids sur l'intervalle [0,1] ;
2. Choix d'un exemple en entrée (X, D) ;
3. Propagation de calcul de cette entrée à travers le réseau ;
4. Calcul de l'erreur quadratique pour l'exemple :

$$E = \frac{1}{2}\sum_{i=1}^{p}(d_i - s_i)^2$$

5. Si erreur en sortie alors pour tous les neurones i (depuis la sortie jusqu'à l'entrée), nous calculons le gradient a_i comme suit :

   Si i est l'indice d'un neurone de sortie alors :

$$a_i = f'(x_i).(d_i - s_i)$$

   Si i est l'indice d'un neurone caché ou d'entrée alors :

$$a_i = f'(x_i)\sum_{k=1}^{N^{C+1}} a_k W_{ki}$$

   Avec N^{c+1} est le nombre de neurones de la couche $c+1$ (en amont de la couche actuelle).

6. Correction des poids des neurones pour atténuer l'erreur :

$$W_{ij}(t+1) = W_{ij}(t) + \alpha.a_i.s_j$$

7. Retour à l'étape 2 que l'on répète jusqu'à atteindre un seuil minimal d'erreur ou un nombre maximal d'itérations.

3. 3 Clustering

Le Clustering est le processus qui consiste à regrouper les individus d'une population dans un nombre limité de groupes (segments, clusters et ou partitions), qui ont deux

propriétés. D'une part, ils ne sont pas prédéfinis, mais découvertes au cours de processus. D'autre part, les clusters regroupent les individus ayant des caractéristiques similaires et séparent les individus ayant des caractéristiques différentes (homogénéité interne et hétérogénéité externe), qui peuvent être mesurées par des critères traduisant la ressemblance (ou la dissemblance) entre individus [Tuf 2003]. Le problème de Clustering est très ancien et populaire dans plusieurs domaines. Il est traité à travers plusieurs ouvrages dont : Benzécri (1973) [Ben 1973], Caillez et Pages (1976) [Cai 1976], Roux (1986) [Rou 1986], Celeux et col. (1989) [Cel 1989].

Dans cette section, nous présentons une définition formelle pour le problème de Clustering suivie d'une classification des différents algorithmes de Clustering existants. Puisque la construction des clusters est basée sur des mesures de ressemblances (ou de dissemblances) entre individus, la section 3. 3. 3 est consacrée pour décrire les mesures les plus reconnues. Finalement, nous détaillons quelques techniques de Clustering.

3. 3. 1 Définition formelle

Une formulation mathématique de la définition de Clustering peut être la suivante :

Soient D = { X_1, X_2, …, X_n } un ensemble de *n* individus tirés de la population d'examen ; Chaque individu X_i (i : 1, 2, …, *n*) est d'écrit par *P* attributs a_j dans un espace de *P*-dimensions (j : 1, 2, …, P) comme suit : X_i = [X_i^1, X_i^2, …, X_i^P], telle que X_i^j représente la valeur de l'attribut a_j pour l'individu X_i ; L'objectif de Clustering est de partitionner D en *k* (*k* est un entier positive) clusters C_α (α : 1, …, *k*).

$$D = C_1 \cup C_2 \cup ... \cup C_k \cup C_{ouliers} \ , \ C_i \cap C_j = \phi \text{ pour } 1 < i, j < k \text{ et } i \neq j$$

La recherche de ces clusters est généralement guidée par la maximisation d'une fonction de similarité inter-clusters et la maximisation d'une fonction de dissimilarité intra-clusters.

3. 3. 2 Classification des algorithmes de Clustering

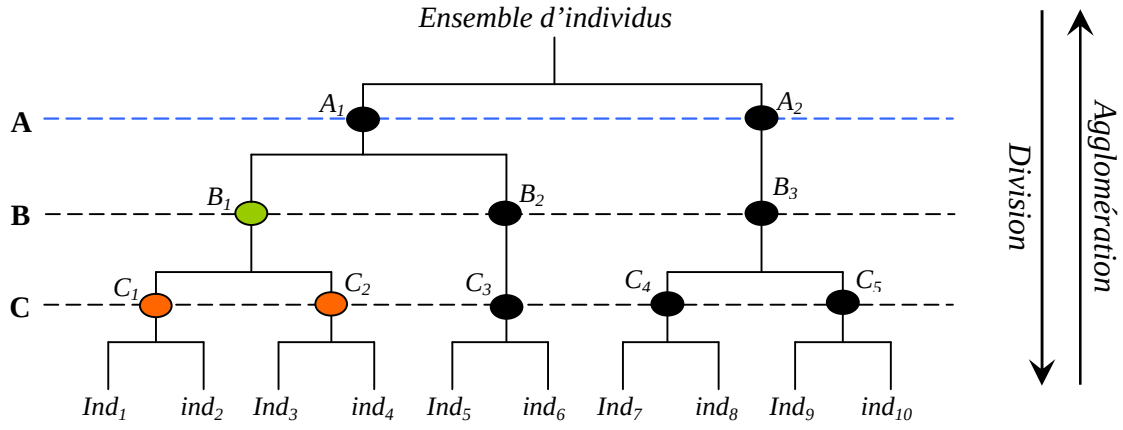
En se basant sur la façon dont les clusters sont construits, les différents algorithmes de Clustering sont traditionnellement groupés en trois classes de méthodes suivantes [Jai 1988] [Jai 1999] [Ber 2002] :

3. 3. 2. 1 Méthodes par partitionnement

Elles consistent à partitionner l'ensemble d'individus en plusieurs partitions ou clusters, qui sont construites en regroupant les individus autour de noyaux choisis initialement au hasard puis elles améliorent itérativement ces partitions initiales on se basant sur une fonction de coût à minimiser. Par exemple, minimiser une fonction de distance entre individus. On distingue, dans cette famille de méthodes, deux variantes : les algorithmes de classification autour de centres mobiles (*Forgy* [For 1965], *k-means* [Mac 1967], *nuées dynamiques* [Did 1971], *k-modes* [Hua 1997a], *k-prototypes* [Hua 1997b]) et les algorithmes de classification autour d'objets représentatifs (PAM « *Partitioning Around Medoids* » [Kau 1990], CLARA « *Clustering LARge Applications* » [Kau 1990], CLARANS « *Clustering Large Applications based on RANdomized Search* » [Ng 1994]). Pour expliquer le principe de fonctionnement des méthodes par partitionnement, nous détaillons dans la section 3. 3. 4 l'algorithme le plus utilisé dans cette famille, qu'est l'algorithme *k-means*.

3. 3. 2. 2 Méthodes hiérarchiques

Contrairement aux méthodes par partitionnement, les méthodes hiérarchiques ne produisent pas un seule partitionnement mais un arbre hiérarchique de partitions, appelé *dendrogramme*. Chaque coupure, aux différents niveaux de dendrogramme, correspond à un partitionnement. Par exemple, sur la figure 3. 10, la coupure A donne deux partitions A_1 et A_2.



$\boldsymbol{A_1}$= {ind_1, ind_2, ind_3, ind_4, ind_5, ind_6}, $\boldsymbol{A_2}$= {ind_7, ind_8, ind_9, ind_{10}}
$\boldsymbol{B_1}$= {ind_1, ind_2, ind_3, ind_4}, $\boldsymbol{B_2}$= {ind_5, ind_6}, $\boldsymbol{B_3}$= {ind_7, ind_8, ind_9, ind_{10}}
$\boldsymbol{C_1}$= {ind_1, ind_2}, $\boldsymbol{C_2}$= {ind_3, ind_4}, $\boldsymbol{C_3}$= { ind_5, ind_6}, $\boldsymbol{C_4}$= { ind_7, ind_8}, $\boldsymbol{C_5}$= { ind_9, ind_{10}}

Figure 3. 10 Dendrogramme

Alors chaque nœud de dendrogramme correspond à un cluster, qui peut être construit soit par division d'un nœud de haut niveau soit par agglomération des nœuds de bas niveau. Par exemple, sur la figure précédente le cluster B_1 est l'agrégation de deux clusters C_1et C_2 ou bien, les deux clusters C_1 et C_2 et le résultat de la division de cluster B_1. En effet, selon la

façon dont le dendrogramme est construit, agglomérative (respectivement ascendante) ou divisive (respectivement descendante), cette famille de méthodes est subdivisée en deux sous classes :

- *Méthodes agglomératives*: Construisent un dendrogramme par groupement des clusters similaires. Au départ, les clusters contiennent un seul individu ($k = n$, tel que k : le nombre de clusters et n : le nombre d'individus dans l'ensemble d'individus). On se basant sur des mesures d'agrégation des clusters, appelées *critères d'agrégation*, ces derniers sont groupés deux à deux pour construire des clusters de plus grande taille pour le niveau supérieur de dendrogramme. Finalement, le processus d'agglomération des clusters est terminé à l'aboutissement d'un seul cluster contenant tout les individus de l'ensemble d'individus. Parmi les algorithmes les plus fréquemment utilisés dans cette famille nous citons : AGNES « *AGglomerative NESting* » [Kau 1990], CURE « *Clustering Using REpresentatives* » [Guh 1998], CHAMELEON [Kar 1999].
- *Méthodes divisives*: À l'inverse aux méthodes agglomératives, les méthodes divisives construisent le dendrogramme par des divisions successives des clusters on se basant sur des mesures de dissimilarités entre les individus. Débutant par un seul cluster contenant tous les individus de l'ensemble d'individus. À chaque niveau de dendrogramme une évaluation de dissimilarité entre les individus de même cluster est réalisée ; on se basant sur cette dissimilarité entre individus, chaque cluster est divisé en deux clusters de telle sorte à maximisée la dissimilarité intra-clusters. À l'aboutissement des clusters à un seul individu le processus de division successive des clusters est achevé. Pratiquement cette famille de méthodes est moins utilisable par comparaison aux méthodes agglomératives. Cependant, DIANA « *DIvisive ANAlysis* » [Kau 1990] et MONA « *MONothetic Analysis* » [Kau 1990] sont les algorithmes les plus fréquemment rencontrés.

La famille d'algorithmes hiérarchiques construit un dendrogramme de la racine aux feuilles. Cependant, plusieurs algorithmes de cette famille empêchent la construction complète de ce dendrogramme par la spécification des conditions d'arrêts. Par exemple, spécifié le nombre k de clusters souhaités ou atteindre un seuil maximum pour une fonction de similarité (ou de dissimilarité) entre individus. Dans la section 3. 3. 4, nous expliquant le principe de fonctionnement de cette famille d'algorithmes en décrivant l'algorithme de la *Classification Hiérarchique Ascendante (CHA).*

3. 3. 2. 3 Méthodes par densité

Basant sur la notion de densité, de connectivité et de frontière, ces méthodes permettre de grouper les individus situés dans des zones de forte densité. Par opposition aux méthodes

par partitionnement et hiérarchiques, qui nécessitent la spécification de nombre de clusters et produisent des clusters sous formes convexes, les méthodes à base de densité peuvent découvrir des clusters avec des formes arbitraires et ils n'ont pas besoin de fixer a priori le nombre de clusters. Tel qu'un cluster, défini comme une composante dense connectée, peut s'accroître dans n'importe quelle direction selon que la densité le dirige. La figure 3. 11 illustre quelques formes de clusters qui présentent un problème pour les méthodes par partitionnement (par exemple, *k-means*), mais manipulés correctement par les méthodes à base de densité.

Figure 3. 11 Quelques formes de clusters difficiles à manipuler par *k*-means

Cette famille de méthodes comprend DBSCAN « Density-Based Spatial Clustering of Applications with Noise » [Est 1996], OPTICS « Ordering Points To Identify Clustering Structure » [Ank 1999], DENCLUE « DENsty-based CLUstEring » [Hin 1998].

3. 3. 3 Similarité, Dissimilarité et Distance

Toutes les méthodes de Clustering citées précédemment utilisent des notions de similarité et de dissimilarité pour qualifier par une grandeur numérique la ressemblance entre deux individus. Elles peuvent aussi utiliser la notion de distance qui est la plus contraignante de la dissimilarité.

Il n'y a pas une définition unique pour la similarité entre individus, cette notion dépend de type de données manipulées ainsi que de type de la similarité recherchée. Dans cette section, nous décrivons premièrement les propriétés de ces mesures (similarité, dissimilarité et distance), en suite nous exposant certaines mesures concernant quelques type de données (quantitatives, binaires, qualitatives).

Propriétés des notions de similarité, dissimilarité et distance

Toutes ces notions sont des mesures de dissemblances (ou de ressemblances) entre individus. Une fonction de similarité « *sim* » (resp. dissimilarité « *dis* », distance « *d* ») est une application de $D \times D$ à valeurs positives ou nulles. Telle que D est un ensemble de n d'individus X_i (voir la section 3. 3. 1). Ces fonctions obéissent aux propriétés suivantes:

- La positivité

$$\forall x, y \qquad sim(x, y) \geq 0$$
$$dis(x, y) \geq 0$$
$$d(x, y) \geq 0$$

- La Symétrie

$$\forall x, y \qquad sim(x, y) = sim(y, x)$$
$$dis(x, y) = dis(y, x)$$
$$d(x, y) = d(y, x)$$

- La distance vérifie l'inégalité triangulaire :

$$\forall x, y, z \qquad d(x, z) \leq d(x, y) + d(y, z)$$

- La distance entre un individu et lui-même vaut 0 :

$$\forall x, y \qquad d(x, y) = 0 \quad \text{Si et seulement si} \quad x = y$$

- La similarité vérifie que la similarité maximum k est atteinte seulement pour la similarité entre un individu et lui-même :

$$\forall x, y, x \neq y \qquad k = sim(x, x) = sim(y, y)$$
$$k > sim(x, y)$$

- La dissimilarité vérifie que la dissimilarité minimum est atteinte seulement pour la dissimilarité entre un individu et lui-même et qu'elle vaut 0 :

$$\forall x, y, x \neq y \qquad 0 = dis(x, x) = dis(y, y)$$
$$0 < dis(x, y)$$

- Lorsque la distance ne satisfait pas à l'inégalité triangulaire, on parle alors de distance semi-métrique. On appelle distance ultra-métrique une distance qui vérifie en outre :

$$\forall x, y, z \qquad d(x, y) \leq \max(d(x, z), d(z, y))$$

Distances pour les données quantitatives

Soient $X_i = [\, X_i^{\,1}, X_i^{\,2}, \ldots, X_i^{\,P} \,]$ et $Y_j = [\, Y_j^{\,1}, Y_J^{\,2}, \ldots, Y_J^{\,P} \,]$ deux individus de l'ensemble D. Les distances les plus utilisées pour les données quantitatives sont :

- *La distance Euclidienne* qui calcule la racine carrée de la somme des différences carrées entre les coordonnées de deux individus X_i et Y_j :

$$d(X_i, Y_j) = \sqrt{\sum_{k=1}^{P} (X_i^k - Y_j^k)^2}$$

- *La distance Manhattan* qui calcule la somme des valeurs absolues des différences entre les coordonnées de deux individus X_i et Y_j :

$$d(X_i, Y_j) = \sum_{k=1}^{P} \left| X_i^k - Y_j^k \right|$$

- *La distance de Minkowski* qui est une métrique de distance générale. Elle s'écrit :

$$d(X_i, Y_j) = \sqrt[q]{\sum_{k=1}^{P} \left| X_i^k - Y_j^k \right|^q}$$

Où q est un entier naturel non nul. On note que le cas $q = 1$ correspond à la distance de Manhattan. La distance Euclidienne correspond aussi à la distance de Minkowski quand $q = 2$.

Dissimilarités pour les données binaires

La première méthode consiste à considérer les données binaires comme des données quantitatives et donc d'utiliser les distances décrites précédemment. Une autre méthode est l'utilisation d'indices ou de coefficients concernant spécifiquement les données binaires.

Soient X_i *et* Y_j deux individus de l'ensemble *D*. Tous les indices, concernant les données binaires, interviennent les quatre nombres suivants :

a : nombres de caractéristiques communes à X_i et Y_j.

b : nombres de caractéristiques possédés par X_i et non par Y_j.

c : nombres de caractéristiques possédées par Y_j et non par X_i.

d : nombres de caractéristiques que ne possèdent ni X_i ni Y_j.

Nous citons à titre d'exemple les indices suivants :

- Le coefficient de correspondance simple

$$d(X_i, Y_j) = \frac{b + c}{a + b + c + d}$$

- Le coefficient de Jaccord

$$d(X_i, Y_j) = \frac{b + c}{a + b + c}$$

- L'indice de Russel et Rao

$$d(X_i, Y_j) = \frac{a}{a+b+c+d}$$

Dissimilarités pour les données qualitatives

Une première méthode consiste à créer un codage disjonctif créant autant de variables binaires qu'il y a de modalités. Ensuite, il suffit d'utiliser les distances pour les données binaires ou quantitatives.

Supposons que nous avons les individus X_i et X_j :

	Couleur	***Présence***
X_i	*Rouge*	*Oui*
X_j	*Vert*	*Oui*

Dom (couleur)= {Rouge, Vert, Bleu}
Dom (présence)= {Oui, Non}

Figure 3. 12 Exemple de données qualitatives

Le codage disjonctif crée donc cinq variables binaires qui sont : « Couleur = Rouge », « Couleur = Vert », « Couleur = Bleu », « Présence = Oui », « Présence = Non ». Comme il montre la figure suivante :

	Couleur=Rouge	***Couleur=Vert***	***Couleur=Bleu***	***Présence=Oui***	***Présence=Non***
X_i	*1*	*0*	*0*	*1*	*0*
X_j	*0*	*1*	*0*	*1*	*0*

Figure 3. 13 Exemple de disjonction de données qualitatives

Soient, m : Le nombre de correspondances entre X_i et Y_j ;

P : Le nombre de variables binaires ;

Nous avons ainsi : $dis(X_i, Y_j) = \frac{P-m}{p} = \frac{5-3}{5} = \frac{2}{5} = 0{,}5$

Une autre méthode consiste à utiliser des distances spécifiques aux variables qualitatives. La plus utilisée est le coefficient d'appariement qui donne le pourcentage de valeurs différentes, elle est exprimée par la formule suivante :

$$\forall\ X_i, Y_j \qquad dis(X_i, Y_j) = \frac{1}{P}\sum_{K=1}^{P} match(X_i^K \neq Y_j^K)$$

Nous avons ainsi :

$$dis(X_i, Y_j) = \frac{1}{2}[match(X_i^{Couleur} \neq Y_j^{Couleur}) + match(X_i^{Présence} \neq Y_j^{Présence})] = \frac{1}{2}(1+0) = 0{,}5$$

Dissimilarité pour les données mixtes

Dans le cas de données mixtes, les valeurs des attributs sont de natures différentes (qualitative, quantitative, binaire). Le but est de trouver une distance s'appliquant sur ces données. Il existe de nombreuses techniques permettant de définir une dissimilarité pour des données mixtes [Maz 2002].

La première méthode consiste à transformer les données quantitatives en données qualitatives par le principe de discrétisation, et une autre méthode consiste à transformer les données qualitatives en données quantitatives en créant une disjonction. Enfin, il est possible d'utiliser des distances s'appliquant aux variables mixtes. La distance la plus utilisées est :

$$\forall X_i, Y_j \qquad dis(X_i, Y_j) = \frac{1}{P}\sum_{K=1}^{P} contribution(X_i^K, Y_j^K)$$

Telle que :

$$contribution(X_i^K, Y_j^K) = \begin{cases} 0 & \text{Si } a_k \text{ est une donnée qualitative et } X_i^K = Y_j^K \\ 1 & \text{Si } a_k \text{ est une donnée quantitative et } X_i^K \neq Y_j^K \\ \left|X_i^k - Y_j^k\right| & \text{Si } a_k \text{ est une donnée quantitative normalisée} \end{cases}$$

Similarité et dissimilarité entre groupes de données

Certaines méthodes de Clustering ont besoin de déterminer la dissemblance entre des groupes d'individus. Il convient donc de demander sur quelle base peut-on calculer une dissemblance entre un individu et un groupe et par la suite une distance entre deux groupes. Ceci revient à définir une stratégie de regroupement des éléments, c'est-à-dire, se fixer des règles de calcul des distances entre groupements disjoints d'individus, appelées *critères d'agrégation*. Cette distance entre groupements pourra en général se calculer directement à partir des distances des différents éléments impliqués dans le regroupement.

Soient X, Y, Z trois objets et d une mesure de dissimilarité. Par exemple, si on regroupe les objets X et Y en un seul élément noté H, on peut définir la distance de ce groupement à Z on utilisant différents critères d'agrégation. Parmi les plus utilisés on cite [Leb 1998] :

- *Critère du Saut Minimal (« Single Linkage » ou « Nearest Neighbour »)* : La distance entre parties est la plus petite distance entre éléments des deux parties :

$$d(H, Z) = \min \{d(X, Z), d(Y, Z)\}$$

- *Critère du Saut Maximal (« Complete Linkage » ou « Furthest Neighbor »)* : Ce critère, appelé également diamètre, est défini en prenant la plus grande distance entre les deux parties :

$$d(H, Z) = \max \{d(X, Z), d(Y, Z)\}$$

- *Critère du Saut Moyen (« Beverage » ou « Between-groups Linkage »)* : Cette méthode fournit un compromis entre les deux premiers critères :

$$d(H, Z) = \frac{n_x d(X, Z) + n_y d(Y, Z)}{(n_x + n_y)}$$

Où n_x et n_y sont les effectifs des deux groupes *X* et *Y*.

3. 3. 4 Exemples d'algorithmes de Clustering

3. 3. 4. 1 *k*-means

L'algorithme *k*-means [Mac 1967], appelé également *k*-moyenne, est le plus populaire des méthodes de Clustering citées précédemment. Puisque il est facile à implémenter et de complexité polynomiale, *O (n)*, *k*-means est très utilisés pour résoudre le problème de segmentation dans plusieurs applications scientifiques et industrielles. Son nom vient de la représentation des clusters par leurs centres, d'où l'appellation « *Centroïdes* ».

Soient *D* un ensemble de *n* individus X_i (voir la section 3. 3. 1). L'algorithme *k*-means, consiste à partitionner l'ensemble *D* en *k* clusters. Son principe de fonctionnement se décompose en quatre étapes suivantes :

1. *Initialisation*
   - Spécifier le nombre de clusters *k* ;
   - Sélectionner *k* points m_j $(j:1,...,k)$ dans l'ensemble d'individus *D*. Le choix de ces points s'effectue soit par un fonction aléatoire soit selon quelques heuristiques spécifiées par les experts de domaine. Les m_j points sélectionner sont les centres initiaux des *k* clusters à construire ;
   - Spécifier une fonction de distance à optimiser. La fonction la plus utilisée est la distance euclidienne.

- Le résultat de l'algorithme est un ensemble C de k clusters, telle que $C = \{C_1, C_2, ..., C_k\}$.

2. *Affectation*

- Calculer la distance entre chaque individu X_i et les centres m_j :

$$\forall\ i : \overline{1,n},\ \forall\ j : \overline{1,k} \qquad d(X_i, m_j) = \sqrt{\sum_{l=1}^{P} (X_i^l - m_j^l)^2}$$

- Une fois les distances sont calculées, affecter chaque individu X_i au cluster C_j dont le centre m_j est le plus proche à X_i. Donc, on cherche à minimiser la distance « $d(X_i, m_j)$ ».

3. *Recalculer les centres*

- Pour chaque cluster C_j, calculer son nouveau centre m_j. La fonction la plus utilisée pour mettre à jour les centres des clusters est la « Moyenne » des éléments du chaque cluster.

4. *Réaffectation*

- Aller à l'étape (2) et refaire la tâche de l'affectation en utilisant les nouveaux centres m_j, en suite refaire les tâches (3) et (4) dans cette ordre.

L'algorithme *k*-means s'arrêt à la stabilisation des clusters, c'est-à-dire lorsque le nombre d'individus qui change de clusters est très petit.

3. 3. 4. 2 Classification Hiérarchique Ascendante

La Classification Hiérarchique Ascendante (CHA) permet la création d'un arbre binaire de partitions, appelé dendrogramme (voir section 3. 3. 2. 2). La construction d'un tel arbre débute par un ensemble de clusters contenant chacun un seul individu. Ensuite, le processus de construction de l'arbre regroupe les clusters deux à deux, selon un *critère d'agrégation des clusters*, jusqu'à l'arrivé à un seul cluster contenant tous les individus de l'ensemble d'individus. Ce principe de fonctionnement peut être traduit par l'algorithme suivant :

Algorithme 3. 3 Classification Hiérarchique Ascendante (CHA)

Données : - $D = \{ X_1, X_2, \ldots, X_n \}$; *// voir section 3. 3. 1*

- $d(C_i, C_j)$: Une mesure de similarité ou de dissimilarité entre les clusters C_i et C_j ; *// Critère d'agrégation des clusters*

Sorties : Une hiérarchie de partitions $\{P_0, P_1, \ldots, P_{n-1}\}$ de D ;

Début

Initialisation : $P_0 = \{C_1, C_2, \ldots, C_n\}$, tel que $\forall\ j : 1, \ldots, n \quad C_j = \{X_j\}$;

Pour $k = 1$ à $(n-1)$ **faire**

Pour chaque couple (C_i, C_j) de P_{k-1} **faire**

Choisir $C_{opt} = d(C_i, C_j)$ tel que

Fin pour ;

$C_{fucion} = C_i \cup C_j$;

$P_k = (P_{k-1} - \{C_i, C_j\}) \cup \{C_{fucion}\}$;

Fin pour ;

Fin.

3. 3. 4. 3 Cartes topologiques de Kohonen

Les réseaux de Kohonen, ou les cartes topologiques de Kohonen, font partie de la famille des modèles dits à apprentissage non supervisé. Puisque les réseaux de Kohonen « s'auto-organisent » autour des données, il sont souvent connu sous l'appellation *cartes topologiques auto-organisatrices* (*Self-Organizing Maps – SOM)*. Il se décompose d'un ensemble de nœuds répartis en deux couches, couche d'entrée et couche cachée, et des connexions entre ces nœuds (figure 3. 14). A la différence des Perceptron MultiCouche (PMC), étudié dans la section 3. 2. 2. 4, les cartes topologiques de Kohonen ne possèdent pas la notion de couche de sortie, puisqu'il n'y a pas de variables à prédire ; son but est d'apprendre la structure des données pour pouvoir y distinguer des segments (ou des clusters).

- La couche d'entrée est formée par les vecteurs d'entrée, avec un nœud pour chacune des P variables utilisées dans le Clustering.
- La couche cachée, qui est la couche de Kohonen.

  Les nœuds de cette couche sont répartis uniformément sur une grille généralement rectangulaire de $l \times m$ nœuds, et chacun de ces $(l.m)$ nœuds est relié à chacun des P nœuds de la couche d'entrée, avec un certain poids W_{ijk} ($i \in [1, l]$, $j \in [1, m]$, $k \in [1, P]$).

  Les nœuds de la couche cachée ne sont pas connectés entre eux.
- Une distance euclidienne est utilisée comme une fonction de transfert.

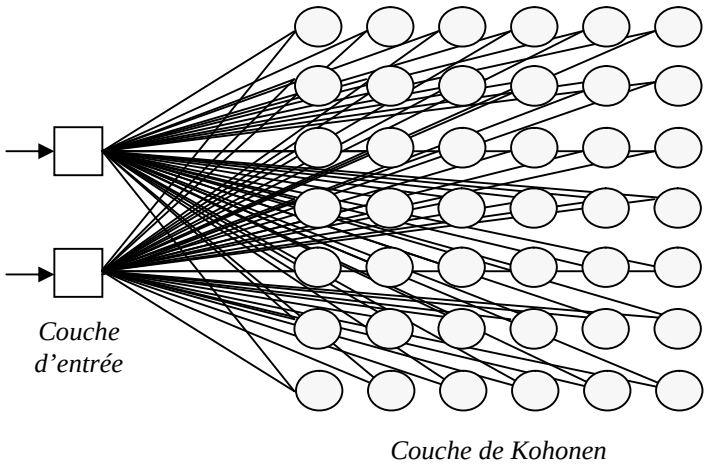


Figure 3. 14 Réseau de Kohonen

L'algorithme d'apprentissage des réseaux de Kohonen est de type compétitif. L'apprentissage compétitif, comme son nom indique, consiste à faire compétitionner les neurones du réseau pour déterminer lequel sera actif à un instant donné. Il produit un vainqueur ainsi que, parfois, un ensemble de neurones voisins du vainqueur, et l'adaptation des poids du réseau est limitée à ce vainqueur et, potentiellement, son voisinage. L'effet de l'apprentissage compétitif est de renforcer la sensibilité des neurones vainqueurs aux patrons d'entrée qui leur ont permet de gagner, et ce, aux dépens des autres patrons. Cependant, plutôt que d'entraîner uniquement le neurone vainqueur, tous les neurones dans le voisinage étendu du vainqueur ont le droit à un apprentissage. En principe, plus un neurone est loin du vainqueur plus faible sera sa correction.

La taille du voisinage diminue généralement au cours de l'apprentissage. Au début, le voisinage peut être la grille tout entière et à la fin, le voisinage peut être réduit au nœud lui-même [Tuf 2003].

Le principe de l'algorithme d'apprentissage des réseaux de Kohonen peut être résumé comme suit :

1. Initialiser aléatoirement les poids synaptiques W_{ijk} suivant la topologie choisie (structure et taille de la carte) ;
2. Répéter
   - Présenter un individu $(x_k)_{1\le k\le P}$;
   - Calculer les distances euclidiennes séparant (x_k) des $(l.m)$ nœuds de la couche de Kohonen :

$$d_{ij}(x) = \sum_{k=1}^{P} (x_k - p_{ijk})^2$$

- Retenir le neurone vainqueur qui va représenter l'individu (x_k) de sorte que $d_{ij}(x)$ soit minimum ;
- Ajuster les poids du neurone vainqueur et de ses voisins en utilisant la règle d'apprentissage suivante :

$$p_{ijk}^{t} = p_{ijk}^{t-1} + \alpha(x_k - p_{ijk}^{t-1}) \ ,$$

Où α représente le taux d'apprentissage ;

Jusqu'à ce que tous les individus de l'ensemble d'apprentissage ont été présentés au réseau et que tous les poids ont été ajustés.

Nous constatons que l'algorithme de Kohonen est assez proche de la méthode d'agrégation des centres mobiles, puisque chaque individu présenté en entrée est représenté par un nœud du réseau qui lui est le plus proche au sens de la distance euclidienne. Ce nœud sera le segment de l'individu [Tuf 2003].

3. 4 Règles d'association

L'extraction des connaissances à partir des bases de données transactionnelles se réfère souvent au problème de recherche des règles d'association, qui a fait l'objet de nombreux travaux de recherche ces dernières années [Agr 1993] [Agr 1994] [Man 1994a] [Man 1994b] [Hou 1995] [Sav 1995] [Sri 1995] [Soo 1995] [Bri 1997]. Il consiste à extraire des corrélations ou des rapports d'association intéressants entre les objets d'une grande base de données transactionnelle. Introduit par [Agr 1993], le problème de recherche des règles d'association tire ces origines à partir de problème de panier de la ménager, « *Market Basket Problem* », il peut être résumé ainsi : étant donné une base de données de transactions (les paniers), chacune composée d'items (les produits achetés), la découverte d'associations consiste à chercher des ensembles d'items, fréquemment liés dans une même transaction, ainsi que des règles les combinant. Un exemple d'association pourrait révéler que « 75% des gens qui achètent de la bière, achètent également des couches ». Actuellement le problème de recherche des règles d'association est étendu aux différents types de base de données (relationnelles, dimensionnelles, etc.) et concerne un grand champ d'applications, citant la conception des catalogues, les prévisions financières, les politiques de marketing, les diagnostique médicale et de nombreuses d'autres applications. Ainsi, la popularité des règles d'association est due à leurs simplicités.

Selon le niveau d'abstraction, la dimensionnalité et le type des données manipulés, il existe différents types de règles d'association [Han 2000]. Dans ce qui suit, nous se limitons aux règles d'association uni-dimensionnelle, uni-niveau et booléennes. Ainsi, cette section débute par une définition formelle pour le problème de recherche des règles d'association.

Ensuite, nous expliquons brièvement le principe de l'algorithme Apriori qui demeure toujours un algorithme central qui se retrouve dans la plupart des algorithmes consacrés aux règles d'associations.

3. 4. 1 Définition formelle

Formellement, le problème de recherche des règles d'association, tel qu'il est présenté dans [Agr 1993], est défini de la façon suivante :

Soit $I= \{i_1, i_2, \ldots, i_m\}$ un ensemble d'attributs, appelé items. Un ensemble d'items X de cardinalité k ($X \subseteq I$ avec $|X| = k$) est appelé un k-itemsets ou tout simplement itemset. Soit D un ensemble d'itemset, où chaque itemset $t \in D$ est appelé transaction. A Chaque transaction t est associée un identificateur unique, appelée *TID* (*Transaction IDentification*). Nous disons qu'une transaction t contient X, un ensemble d'items, si $X \subset t$. Donc, une règle d'association est une implication de la forme : $X \Rightarrow Y$ [σ ; φ] tels que X est l'antécédent de la règle et Y son conséquent : $X \subset I$, $Y \subset I$, $X \cap Y = \phi$ (c'est-à-dire, X et Y sont des ensembles disjointes d'items).

Une règle d'association $X \Rightarrow Y$ [σ ; φ] traduit le fait que si les items de X sont présents dans une transaction t, alors les items de Y le sont avec une certaine probabilité. La qualité d'une règle d'association est évaluée par deux critères : le support σ et la confiance φ. Plus ces deux critères sont très élevé plus la règle d'association est très intéressante.

- Le support σ exprime la fiabilité ou l'utilité d'une règles d'association. C'est une mesure indiquant le pourcentage de transactions $t \in D$ qui vérifient une règle d'association. En désignant par $|X|$ le nombre de transactions comportant l'ensemble X, par $|X \cup Y|$ le nombre de transactions comportant en même temps X et Y, et par N le nombre total de transactions dans la base D. Le support d'une règle d'association peut être défini ainsi :

$$\text{support}\ (X \Rightarrow Y[\sigma;\varphi]) = prob(X \cup Y) = \frac{|X \cup Y|}{N}$$

- La confiance φ mesure la précision ou la validité d'une règle d'association. C'est une mesure indiquant le pourcentage de transactions $t \in D$ qui vérifient le conséquent d'une règle d'association parmi celle qui vérifient l'antécédent. On utilisant la notation introduite ci-dessus, la confiance d'une règle d'association est donnée comme suit :

$$\text{confiance}(X \Rightarrow Y[\sigma;\varphi]) = prob(X / Y) = \frac{|X \cup Y|}{|X|}$$

Ainsi, à partir d'une base de transactions *D*, le problème de recherche des règles d'association consiste à extraire toutes les règles d'association dont le support et la confiance sont supérieurs à des seuils (*minsup* et *minconf*) spécifiés par les utilisateurs ou les experts de domaine.

3. 4. 2 Algorithme Apriori

Introduit par Agrawal et Srikant [Agr 1994], Apriori est un algorithme très puissant d'extraction des règles d'association booléennes. Son nom vient de fait que l'algorithme utilise des connaissances préalables sur les propriétés des itemsets fréquents. Comme tous les algorithmes de recherche des règles d'association, l'algorithme Apriori est décomposé en deux étapes :

La génération des itemsets fréquents

L'objectif de cette étape est d'extraire, à partir de l'ensemble de données, tous les itemsets dont le support dépasse un certain seuil (*minsup*) défini par l'utilisateur. Les itemsets extraites dans cette étape sont appelés les itemsets fréquents.

Pour extraire tous les itemsets fréquents, l'algorithme Apriori utilise une approche de recherche itérative par niveaux (*Level-Wise Research*) d'où les *k*-itemsets sont utilisés pour trouver les (*k*+1)-itemsets. Au début, tous les *1*-itemsets fréquents sont extraite de la base de données. Cet ensemble est dénoté par L_1. En suite, l'ensemble L_1 est utilisé pour trouver L_2, l'ensemble des 2-itemsets fréquents, lequel est utilisé pour trouver L_3, et ainsi de suite. Ce processus itératif est achevé lorsqu'il n'y a plus d'itemsets fréquents à extraire. A chaque itération de l'algorithme, la génération des itemsets fréquents est effectuée selon ces deux étapes :

- *Étape de jointure* : Pour découvrir L_k, un ensemble des *k*-itemsets candidats est généré par la jointure de l'ensemble L_{k-1} avec elle-même. Cet ensemble est dénoté par C_k. L'opération de jointure est optimisée par cette propriété : « deux éléments de l'ensemble L_{k-1} sont joinable si et seulement s'ils partagent *k*-2 items ». Ainsi, la jointure de deux ensembles est donnée comme suit :

$$L_{k-1} \triangleright\triangleleft L_{k-1} = \left\{ A \triangleright\triangleleft B \middle| A, B \in L_{k-1}, \left| A \cap B \right| = k-2 \right\}$$

- *Étape d'élagage* : Pour améliorer les performances de l'algorithme Apriori, une phase d'élagage est exécuté à chaque itération. En effet, l'étape de l'élagage permet de réduire l'espace de recherche par l'utilisation d'une propriété des itemsets fréquents, souvent appelé propriété d'Apriori : « tout sous ensemble d'un itemset fréquents doit être lui aussi fréquent ». Puisque les itemsets non fréquents génèrent automatiquement des

super-itemsets non fréquents, à chaque itération, l'algorithme Apriori sauvegarde seulement les itemsets fréquents.

Donc, l'ensemble L_k est généré à partir de l'ensemble C_k, qui contient tous les *k*-itemsets que se soit fréquents ou non. Pour déterminer l'ensemble des *k*-itemsets fréquents L_k, la base de données est parcourue pour déterminer le support de chaque *k*-itemsets appartenant à C_k, par la suite déduire L_k devient trivial.

Le principe de génération des itemsets fréquents peut être traduit par l'algorithme suivant :

Agorithme 3. 4 Génération des itemsets fréquents ;

L_1 = { 1-itemsets fréquents } ;
$k \leftarrow 2$;
Tant que $L_{k-1} \neq \phi$ **faire**
C_k = Apriori_gen (L_{k-1}) ; // Génération de l'ensemble des *k*-itemsets candidats
Pour toutes instances $t \in D$ **faire**
C_t = Subset (C_k , *t*) ;
Pour tous candidats $c \in C_t$ **faire**
c.count++ ;
Fin Pour ;
Fin Pour ;
$L_k = \{c \in C_k / c.count \geq \text{minsup}\}$;
K++ ;
Fin Tant que ;
Retourne $L = \bigcup_i L_i$;

La déduction des règles d'association

Cette étape consiste à générer, à partir de l'ensemble des itemsets fréquents, des règles d'association dont la confiance est supérieure à un seuil (*minconf*) spécifié par l'utilisateur. Le processus de déduction des règles d'association, à partir de l'ensemble des itemsets fréquents, est constitué de deux étapes suivantes :

- Pour chaque itemsets fréquents f, générer tous ces sous-ensembles non vides.
- Pour chaque sous-ensembles non vide s de f, produire la règle « $s \Rightarrow (s - f)$ » lorsque

$$\frac{\text{supp - count}(f)}{\text{supp - count}(s)} \geq minconf$$ où *minconf* représente la confiance minimale.

L'algorithme Apriori souffre de deux points faibles très coûteux à savoir : un espace de recherche très grand et, à chaque itération, plusieurs parcourt de la base de données sont effectués. Ce qui le rend inapproprié pour les grandes bases de données. Pour remédier à ces deux problèmes, les auteurs de domaine de recherche des règles d'association ont proposées plusieurs techniques d'optimisation (hachage, partitionnement, échantillonnage, etc.). En effet, presque tous les algorithmes proposés dans ce domaine constituent une amélioration pour l'algorithme Apriori on utilisant ces techniques.

3. 5 Conclusion

Dans ce chapitre, Nous avons présenté quelques techniques d'extraction de connaissances à partir de données. Ces techniques sont réparties en trois classes de problèmes : le problème de classification (les *k*-Plus Proches Voisins, les arbres de décision, les réseaux bayésiens et les réseaux de neurones artificiels), le problème de Clustering (*k*-means, la Classification Hiérarchique Ascendante et les cartes topologiques de Kohonen) et le problème de recherche des règles d'association (l'algorithme Apriori). L'ensemble des techniques présentées, dans ce chapitre, est incomplet. En effet, des techniques d'extraction de connaissances importantes ne sont pas étudiées dans ce chapitre, citant la programmation logique inductive, les machines à vecteurs de support (« *Support Vector Machine* ») [Bur 1998], les algorithmes génétiques (« *Genetic Algorithms* »), la recherche des motifs séquentiels (« *Sequential Patterns* ») [Agr 1995b] et bien d'autres.

L'objectif de notre travail est de proposer une solution parallèle et distribuée quant à l'extraction et à la gestion des connaissances. Avant d'entamer ce travail, nous discutons dans le chapitre suivant les différents problèmes liés à l'application des techniques traditionnelles d'extraction de connaissances sur des environnements de calcul parallèle et distribué.

CHAPITRE 4

Problèmes d'Extraction Parallèle et Distribuée de Connaissances

4. 1 Introduction

L'extraction de connaissances à partir des grands volumes de données, éventuellement hétérogènes et distribués, nécessite le recours à des traitements parallèles et distribués. En effet, l'application de techniques traditionnelles d'extraction de connaissances (ou de Data Mining) sur des grands volumes de données engendre un coût de calcul inacceptable, impose le recours à des traitements parallèles et distribués pour améliorer leurs temps de réponse. En outre, les techniques traditionnelles de Data Mining assument des données centralisées (stockées sur un seul support de stockage) et leurs application sur des données hétérogènes et distribuées nécessite la collection et l'intégration de ces données dans un site centrale, souvent appelé entrepôt de données (*Data Warehouse*), mais cette solution s'est avérée inapplicable en raison des limitations en capacité de stockage, en capacité de traitement et ainsi que en capacité de communication (bande passante limitée). Donc, l'émergence des ressources de stockage de données de haute capacité, tels que les entrepôts de données, et de calcul de haute performances, tels que les grilles de calcul (*Grid Computing*), à beaucoup stimulé les chercheurs de domaine de l'ECD à paralléliser et à distribuer les techniques traditionnelles d'extraction de connaissances (ou de Data Mining). Ainsi, une nouvelle branche dans le domaine de l'ECD a été révélée, l'Extraction Parallèle et Distribuée de Connaissances (EPDC), ou encore « *Parallel and Distributed Knowledge Discovery (PDKD)* » en anglais.

Le but de ce chapitre est de discuter les différents problèmes liés à l'application des techniques traditionnelles d'extraction de connaissances sur des environnements de calcul parallèle et distribué. Au début de ce chapitre, nous présentons des définitions liées au concept d'Extraction Parallèle et Distribuée de Connaissances, ainsi que les facteurs les plus importants qui sont à l'origine de développement des solutions parallèles et distribuées d'extraction de connaissances. Par la suite, nous discutons les différents problèmes liés à l'application des techniques traditionnelles de Data Mining sur des environnements de calcul parallèle et distribué. Enfin, nous décrivons les différents composants algorithmiques de conception parallèle et distribuée des techniques d'extraction de connaissances.

4. 2 Définitions

L'Extraction Parallèle et Distribuée de Connaissances, ou encore « *Parallel and Distributed Knowledge Discovery* » en anglais, se réfère à l'application de processus traditionnel d'Extraction de Connaissances à partir de Données (ECD) sur des environnements de calcul parallèle et distribué. Cependant, le Data Mining Parallèle et Distribué, ou encore « *Parallel and Distributed Data Mining* » en anglais, s'intéresse à l'application des techniques traditionnelles de Data Mining sur des tels environnements.

Comme elle montre la figure 4. 1, le Data Mining Distribué s'exécute à deux niveaux [Tso 199] [Fu 2001] [Par 2003]. Au niveau local (*Data Mining Local*), un ou plusieurs algorithmes de Data Mining s'exécutent localement sur chaque site de l'environnement de Data Mining Distribué et un ensemble de connaissances locales est extraites sur chaque site. Au niveau global (*Data Mining Global*), les connaissances extraites localement sur les différents sites de l'environnement de Data Mining Distribué sont combinées afin de découvrir des connaissances globalement utiles et intéressantes.

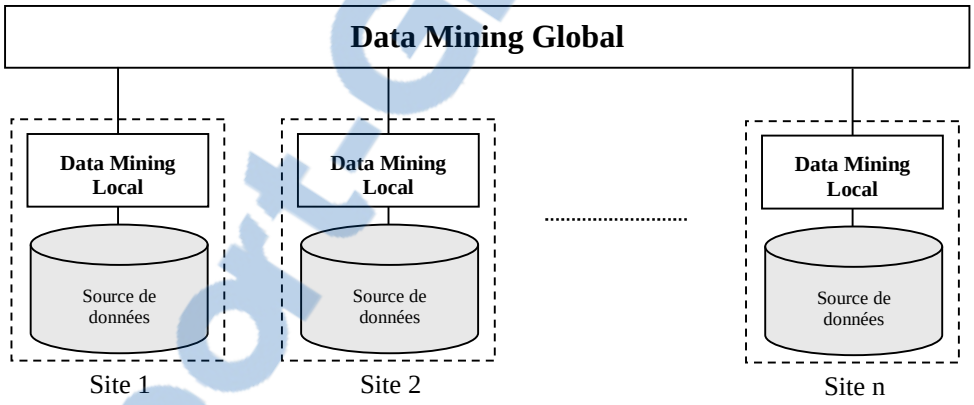


Figure 4. 1 Architecture de Data Mining Distribué

Le Data Mining Distribué est souvent mentionné par le Data Mining Parallèle dans la littérature [Fu 2001]. Bien que tous les deux visent à améliorer les performances des techniques traditionnelles de Data Mining sur des environnements de calcul parallèle et/ ou distribué, ils assument des architectures différentes et adoptent des approches différentes. Le Data Mining Distribué assume un réseau d'ordinateurs ; les données manipulées par les algorithmes de Data Mining Distribué sont distribuées sur les différents sites du réseau et le plus souvent sont des données hétérogènes. Cependant, le Data Mining Parallèle assume une architecture parallèle ; les algorithmes de Data Mining Parallèle manipulent une seule source de données généralement de grande capacité (pas de distribution de données). En effet, l'objectif de Data Mining Distribué est d'extraire des modèles, motifs, règles et toutes autres formes interprétables en connaissances utiles à partir des données distribuées et hétérogènes tandis que, l'objectif de Data Mining Parallèle est d'améliorer les performances des

algorithmes d'extraction de connaissances sur des grandes bases de données en exploitant la puissance de calcul des machine parallèles.

4. 3 Les facteurs d'émergence de Data Mining Parallèle et Distribué

Les facteurs les plus importants qui sont à l'origine de développement des solutions parallèles et distribuées d'extraction de connaissances sont :

Les données sont naturellement distribuées

L'un des principaux facteurs qui ont favorisés le développement des techniques parallèles et distribuées d'extraction de connaissances est la nature des données manipulées, qui sont souvent hétérogènes et distribuées à travers les différents sites d'un environnement de calcul distribué. L'extraction de connaissances à partir de telle catégorie de données nécessite la collection et l'intégration de toutes les données éparpillées sur les différents sites de l'environnement distribué dans un site central, souvent appelé entrepôt de données, et d'appliquer ensuite les techniques traditionnelles de Data Mining. Cependant, même si nous avons suffisamment de capacité de stockage, transférer des grandes quantités de données en un site central peut engendrer un coût de calcul énorme en raison des limitations des réseaux de communication en bande passante et ainsi que en capacité de traitement. En d'autre part, les différents sites de l'environnement distribué englobent souvent des données personnelles et privées qui exigent la protection de leurs confidentialités et ainsi, le déplacement de ces données en un site central n'est pas souhaitable ou inapproprié puisqu'il met leurs confidentialités en danger. En conséquence, il est devenu nécessaire de remplacer les techniques traditionnelles de Data Mining, qui ne peuvent manipulées que des données centralisées, par des techniques parallèles et distribuées qui permettra de traiter les données la où se trouvent sans besoin de les déplacées en site centrale.

Les donnes sont de grand volume

Aujourd'hui, les entreprises ont à leurs dispositions des grandes quantités de données qui augmentent avec les disponibilités de stockage de données. En effet, suite à l'expansion des supports physiques de stockage et les besoins incessants de sauvegarder de plus en plus de données, les techniques traditionnelles de Data Mining qui s'exécutent sur une seule machine se sont avérées inefficaces ; l'application des algorithmes séquentiels de Data Mining sur des grands volumes de données engendre un coût de calcul inacceptable. L'une des solutions exploitées pour réduire le temps de réponse de ces techniques sur les grands volumes de données est l'échantillonnage. Mais, cette solution n'est pas souhaitable puisque la réduction de données entraîne souvent à des modèles (ou connaissances) incorrects. L'autre solution est apparue avec l'émergence des multiprocesseurs et des machines parallèles de hautes

performances, en exploitant ainsi le parallélisme disponible dans les techniques traditionnelles de Data Mining ; l'exploitation de parallélisme améliore considérablement le temps de réponse des techniques d'extraction de connaissances sur les grandes bases de données. Donc, avec l'augmentation des quantités de données disponibles et l'avènement des machines parallèles de hautes performances, la parallélisation des techniques traditionnelles de Data Mining est devenue une tâche impérative.

4. 4 Problèmes d'extraction parallèle et distribuée de connaissances

Fu [Fu 2001] a introduit quelques problèmes liés à l'application des techniques traditionnelles d'extraction de connaissances sur des environnements de calcul parallèle et distribué. Ces problèmes peuvent être énumérés comme suit :

4. 4. 1 Problème d'hétérogénéité de données

L'un des principaux problèmes a considérés en parallélisant et en distribuant les algorithmes de Data Mining est l'hétérogénéité de données. En effet, les données traitées par les algorithmes de Data Mining Parallèle et Distribué peuvent être soit des données homogènes soit des données hétérogènes.

4. 4. 1. 1 Données homogènes

La plupart des algorithmes de Data Mining Parallèle et Distribué assument des données homogènes [Par 2003]. Dans ce cas, les ensembles de données stockés sur les différents sites ont exactement les mêmes attributs, les mêmes domaines d'attributs et partagent le même schéma. La figure ci-dessous illustre un exemple de données homogènes.

Cité	Humidité	Température	Chute de Pluie
San Jose	12%	69° F	0,3 in.
Sacramento	18%	53° F	0.5 in.
Los angeles	86%	72° F	1,2 in.
San Diego	8%	58° F	0 in.

Site 1

Cité	Humidité	Température	Chute de Pluie
Baltimore	10%	23° F	0 in.
Annapolis	13%	43° F	0,2 in.
Bethesda	56%	67° F	1 in.
Glen Burnie	88%	88° F	1,2 in.

Site 2

Figure 4. 2 Exemple de données homogènes

4. 4. 1. 2 Données hétérogènes

Dans le monde réel, les données explorées par les algorithmes de Data Mining Parallèle et Distribué sont souvent hétérogènes. L'extraction de connaissances à partir de telle catégorie de données est une tâche lourde et difficile à résoudre, puisque chaque site comporte un

ensemble d'attributs différent, ainsi que un ensemble de domaines d'attributs différent. En plus, les données hétérogènes sont souvent structurées dans des schémas différents. La figure ci-dessous illustre un exemple de données hétérogènes.

T	Humidité	Température	Chute de Pluie
t_0	10%	23° F	0 in.
t_1	13%	43° F	0.2 in.
t_2	56%	67° F	1 in.
t_3	88%	88° F	1.2 in.

Site 1

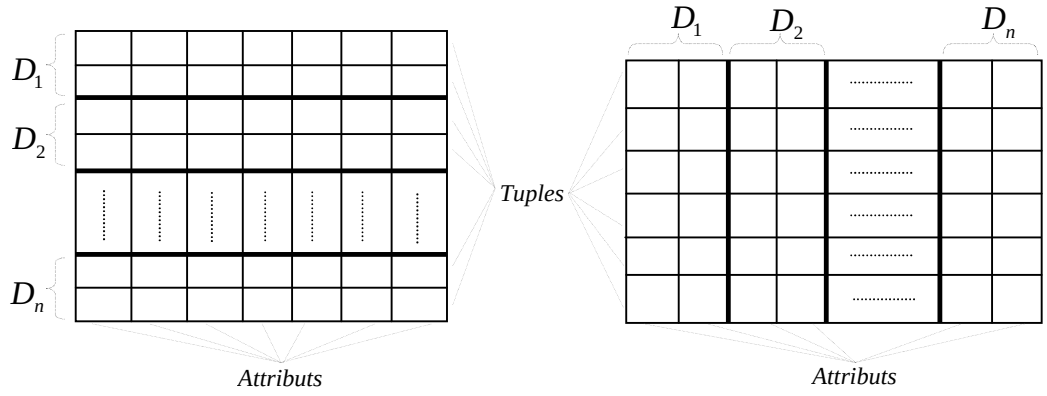
T	Température Corps	Tension
t_0	23° F	60 bpm
t_1	43° F	70 bpm
t_2	67° F	75 bpm
t_3	88° F	80 bpm

Site 2

Figure 4. 3 Exemples de données hétérogènes

4. 4. 2 Problème de fragmentation de données

Dans la plupart des applications informatique, les données sont stockées dans des structures sous forme de tables, où chaque table a N lignes (ou tuples, enregistrements, etc.) et M colonnes (ou attributs, dimensions, etc.). Dans un environnement distribué, les tables de chaque base de données locale peuvent être vues comme des fragments des tables de la base de données globale. Selon la façon dont ces fragments sont construits, il existe généralement deux types de fragmentation de données : fragmentation horizontale et fragmentation verticale.



(a) Fragmentation horizontale (b) Fragmentation verticale

Figure 4. 4 Fragmentation horizontale et fragmentation verticale de données

4. 4. 2. 1 Fragmentation horizontale

La plupart des algorithmes de Data Mining Parallèle et Distribué assument la fragmentation horizontale de données [Fu 2001]. Dans ce cas, toutes les tables de la base de données globale sont découpées horizontalement en plusieurs fragments de telle façon que chaque fragment contient un sous ensemble de tuples d'une table particulière (des fragments

de tuples sont stockés sur les différentes bases de données locales). Ainsi, les bases de données locales partagent les mêmes attributs, ainsi que le même schéma qui est identique au schéma de la base globale. La figure 4. 4 (a) illustre une table de la base de données globale fragmentée horizontalement en n fragments, D_1, D_2, …, D_n.

4. 4. 2. 2 Fragmentation verticale

Dans le cas d'une fragmentation verticale, toutes les tables de la base de données globale sont découpées verticalement en plusieurs fragments de telle façon que chaque fragment contient toutes les tuples d'une table particulière mais pour un sous ensemble d'attributs. Ainsi, chaque base de données locale contient des attributs différents, ainsi que un schéma différent qui un sous schéma de celui de la base de données globale. La figure 4. 4 (b) illustre une table de la base de données globale fragmentée verticalement en n fragments, D_1, D_2, …, D_n.

4. 4. 3 Problème de réplication de données

Pour améliorer les performances des algorithmes de Data Mining Parallèle et Distribué, certaines ou toutes les données (réplication partielle ou totale) sont répliquées sur les différents sites de l'environnement de Data Mining Parallèle et Distribué [Fu 2001]. En effet, la réplication de données permet d'augmenter la disponibilité de données et ensuite d'améliorer les performances en utilisant les copies locales voire les copies plus proches, mais elle introduit le problème de maintenance de la cohérence des copies multiples. Généralement, la réplication de données n'est pas faite pour le but de Data Mining, elle s'agit plutôt d'une décision basée sur les besoins de calcul ou de l'entreprise. Cependant, il est possible de répliquer les données pour le but de Data Mining. Dans ce cas, le décideur doit décider quelles données ou quelle partie de données à répliquer.

4. 4. 4 Coût de communication

Dans les techniques traditionnelles de Data Mining, la préoccupation principale pour améliorer leurs performances est le coût de traitement (temps CPU) et le coût des I/Os (disque ou toutes autres ressources). En revanche, dans les algorithmes de Data Mining Parallèle et Distribué, il faut prendre en considération le coût de communication [Fu 2001]. Ce dernier est déterminé par la largeur de la bande passante du réseau et le nombre de messages envoyés à travers ce réseau. En effet, le coût global d'un algorithme de Data Mining Parallèle et Distribué est en forte relation avec l'infrastructure de l'environnement de calcul parallèle et/ou distribué adopté ; par exemple, pour un réseau lent, le coût de communication diminuera considérablement le coût global de ces algorithmes. Afin de réduire le coût de

communication, plusieurs méthodes de Data Mining Parallèle et Distribué essayent de minimiser le nombre de messages envoyés à travers le réseau, d'autres méthodes utilisent les stratégies d'équilibrage de charge et bien que d'autres méthodes suppose une infrastructure de calcul parallèle et/ ou distribué de haute performance. Par exemple, les grilles de calcul (*Grids Computing*) constituent une infrastructure de calcul distribué très souhaitable pour le Data Mining Distribué.

4. 4. 5 Problème d'intégration des résultats

Pour obtenir des modèles globaux, les algorithmes de Data Mining Parallèle et Distribué combinent et fusionnent les résultats locaux calculés sur les différents sites de l'environnement de calcul parallèle et/ ou distribué adopté. En effet, cette combinaison des résultats n'est pas une simple collection des modèles locaux, puisque un modèle qu'est localement utile et intéressant n'implique pas qu'il est globalement aussi [Fu 2001]. Par exemple, dans la génération parallèle et distribuée de règles d'association, un itemset fréquent sur un site peut être non fréquent globalement. Comme le but de Data Mining Parallèle et Distribué est d'extraire des modèles utiles et intéressants globalement, les modèles et leurs propriétés (exactitude, certitude, etc.) devraient être collectés à partir de tous les sites de l'environnement de calcul parallèle et/ ou distribué et ensuite vérifiés leurs exactitudes globalement.

4. 5 Parallélisation et distribution de Data Mining

L'application des algorithmes traditionnels de Data Mining sur des très grands volumes de données, peuvent être distribués et hétérogènes, nécessite généralement de grandes capacités de traitement qui peuvent être fournies par le recours à des traitements parallèles et distribués. En effet, le Data Mining est bénéficié par l'utilisation des machines parallèles de hautes performances (SMP (*Symmetric Multi-Processors*), Cluster de SMP, MPP (*Massively Parallel Processing*), etc.) et des environnements de calcul distribué à grand échelle (Pair-à-Pair (*Peer-to-Peer* ou *P2P*), grilles de calcul (*Grids Computing*), etc.) pour améliorer ces performances sur les grandes bases de données distribués et hétérogènes, telles que les entrepôts de données repartis. Dans cette section, nous décrivons les différents composants algorithmiques de la conception parallèle et distribuée des algorithmes de Data Mining à savoir, l'architecture parallèle, le type de parallélisme exploité, la stratégie de distribution de données exploité et enfin, la stratégie d'équilibrage de charge adoptée.

4. 5. 1 Les architectures parallèles

Selon le degré de distribution des mémoires, les architectures des machines parallèles peuvent être classifiées en trois grandes catégories [Tan 2008] : les architectures à mémoire partagée, les architectures à mémoire distribuée et les architectures hybrides.

4. 5. 1. 1 Architectures à mémoire partagée

Les architectures à mémoire partagée, appelées également « *Shared-Memory Architectures* » en anglais, sont composées d'un ensemble de processeurs identiques qui accèdent à une même mémoire centrale partagée. La communication entre les processeurs se fait grâce à des lectures/ écritures sur des zones mémoires partagées ce qui simplifie la mise en œuvre de la coopération entre les processeurs. Les architectures à disque partagé, ou encore « *Shared-Disk Architectures* » en anglais, sont très similaires aux architectures à mémoire partagée dans le sens où chaque processeur partage, en plus de la mémoire centrale, le disque. La figure 4. 5 illustre une architecture à mémoire et à disque partagé, dans laquelle chaque processeur dispose de sa propre mémoire cache et partage la mémoire centrale et le disque. En effet, les mémoires caches sont introduites afin d'accélérer les calculs sur chaque processeur en réduisant les accès à la mémoire centrale partagée.

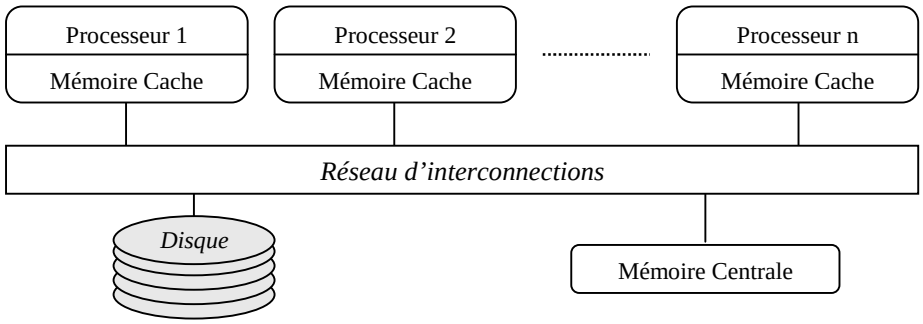


Figure 4. 5 Architectures à mémoire et à disque partagés, « SMP »

Selon le type d'accès à la mémoire partagée par tous les processeurs, les architectures à mémoire et à disque partagé ont d'autres appellations à savoir, les machines de type SMP (*Symmetric Multi-Processors*), les machines de type UMA (*Uniform Memory Access*) et les machines de type NUMA (*Non Uniform Memory Access*).

4. 5. 1. 2 Architecture à mémoire distribuée

Comme elle montre la figure 4. 6, les architectures à mémoire et à disque distribué, ou encore « *Shared-Nothing Architectures* » en anglais, sont constituées d'un ensemble de nœuds, où chaque nœud étant composé d'un processeur, d'une mémoire locale et d'un disque locale. Les nœuds sont connectés entre eux par un réseau d'interconnections au travers duquel ils peuvent coopérer. Grâce au couplage plus faible entre les processeurs et les mémoires, les machines utilisant cette architecture sont plus extensibles à la fois en nombre de processeurs et en capacité mémoire. Cela a permis de construire des machines pouvant avoir plus d'une centaine de processeurs, on parle alors des architectures massivement parallèles (*MPP- Massively Parallel Processing*).

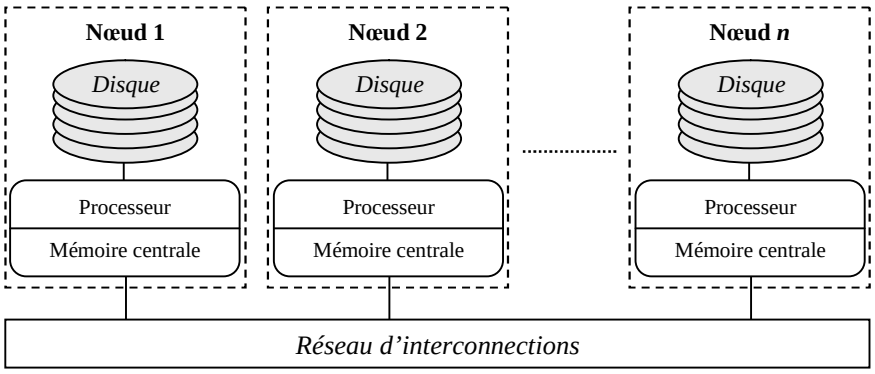


Figure 4. 6 Architectures à mémoire et à disque distribués

La communication entre les processeurs, dans ces architectures, se fait par envoi de messages ce qui rend la mise en œuvre des applications parallèles est plus délicate et la coopération entre les processeurs est plus coûteuse par rapport aux architectures à mémoire partagée.

4. 5. 1. 3 Architectures Hybrides

Les architectures Hybrides, aussi appelées « *Shared-Something Architectures* » en anglais, combinent les deux types d'architectures précédentes, elles sont constituées d'un ensemble de nœuds connectés par un réseau d'interconnections de haut débit et chaque nœud du l'architecture hybride est une architecture à mémoire partagée. Donc, le modèle de mémoire utilisé dans ces architectures est mixte : mémoire partagée au sein d'un même nœud, et mémoire distribuée entre les différents nœuds. La figure 4. 7 illustre l'architecture d'un Cluster de SMPs : une machine à architecture hybride dont laquelle chaque nœud est une machine SMP, et une collection de machines SMP est souvent appelée « *Cluster* ».

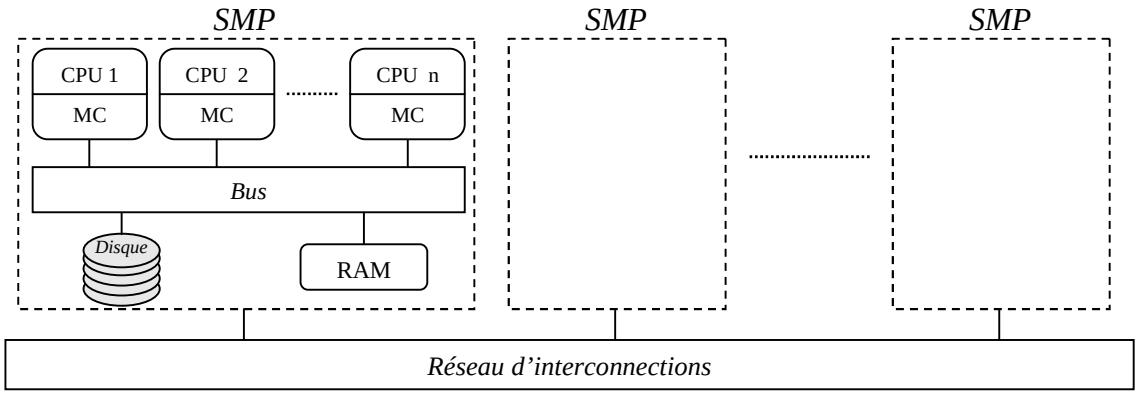


Figure 4. 7 Architectures hybrides, « Cluster de SMPs »

4. 5. 2 Le type de parallélisme

De point de vue de type de parallélisme exploité, la parallélisation et la distribution des algorithmes traditionnels de Data Mining est effectuée suivant deux formes de parallélisme [Fre 1998] [Zak 2000] : le parallélisme de données (partitionnement de la base de données) et le parallélisme de tâches (décomposition de travail en plusieurs tâches). Le parallélisme hybride, qui combine les deux formes de parallélisme, est également possible et peut être souhaitable afin d'exploiter tous le parallélisme disponible dans les algorithmes traditionnels de Data Mining.

4. 5. 2. 1 Parallélisme de données

Le parallélisme de données réfère à l'exécution simultanée d'une même tâche (opérations ou instructions) sur des données différentes. Comme elle montre la figure 4. 8, le parallélisme de données est crée par le partitionnement de la base de données en plusieurs partitions distribuées sur un certain nombre d'éléments de traitement (processeurs ou machines). Les éléments de traitement exécutent simultanément la même tâche sur leurs partitions locales.

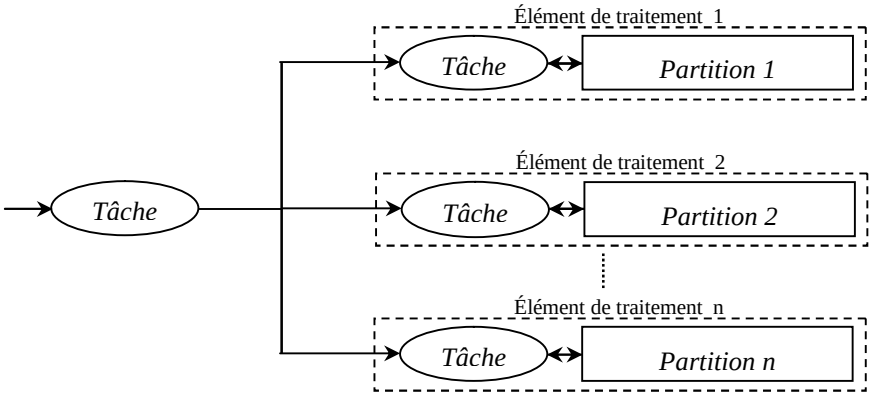


Figure 4. 8 Parallélisme de données

Dans les algorithmes de Data Mining Parallèle et Distribué qui assume le parallélisme de données, chaque élément de traitement reçoit une ou plusieurs partitions de la base de données et exécute le même algorithme séquentiel de Data Mining sur ces partitions locales. En résulte de chaque traitement local sur chaque élément de traitement un ensemble de connaissances locales, lesquelles seront combinées en connaissances globales.

4. 5. 2. 2 Parallélisme de tâches

Contrairement au parallélisme de données, le parallélisme de tâches (ou également le parallélisme de contrôle) réfère à l'exécution concurrente de plusieurs tâches différentes sur les mêmes données. Comme elle montre la figure 4. 9, le parallélisme de tâches nécessite la

décomposition de la tâche globale en plusieurs sous-tâches, tâche 1, tâche 2, …, tâche n, et chaque sous-tâche est assignée à un élément de traitement qu'elle exécutera sur ces données locales.

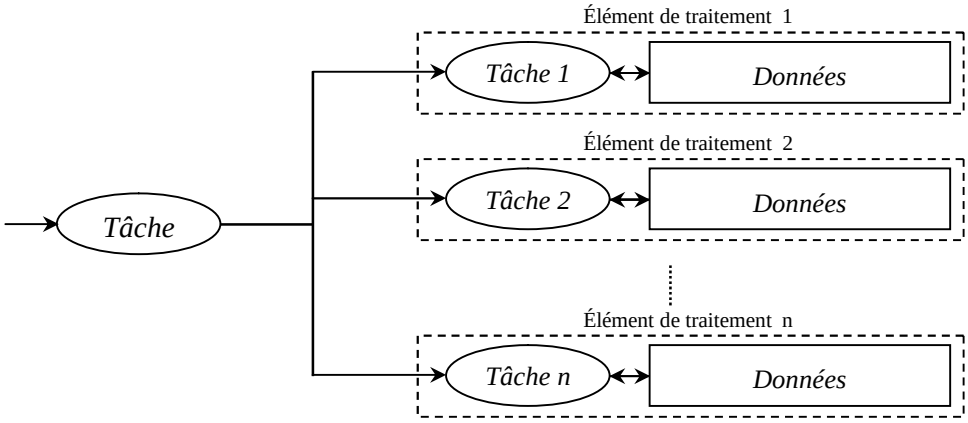


Figure 4. 9 Parallélisme de tâches

Dans le Data Mining Parallèle et Distribué, l'exploitation de parallélisme de tâches est effectuée par la décomposition d'une version séquentielle pour un algorithme de Data Mining en plusieurs sous algorithmes. Chaque élément de traitement de l'environnement de Data Mining parallèle et distribué prend en charge l'exécution d'un sous algorithme afin d'accomplir une partie de la charge de travail de l'algorithme séquentiel. Ainsi, le parallélisme de tâches permet d'améliorer les performances des algorithmes de Data Mining par la distribution de la charge de travail à travers les différents éléments de traitement, mais il engendre des algorithmes de complexité élevée à cause de la forte dépendance entre les sous algorithmes exécutés sur les différents éléments de traitement.

4. 5. 3 Les stratégies de distribution de données

La distribution de données indique la façon de répartition des données entre les différents processeurs de l'architecture parallèle adoptée. Dans la conception parallèle et distribuée des algorithmes de Data Mining, deux stratégies de partitionnement de données sont généralement considérées [Tan 2008] [Fre 1998] : le partitionnement horizontal et le partitionnement vertical. La stratégie de partitionnement vertical permet de partitionner les données verticalement à travers tous les processeurs de telle façon que chaque processeur dispose de toutes les tuples d'une table particulière, mais pour un sous ensemble d'attributs. La stratégie de partitionnement horizontal permet de partitionner les données horizontalement à travers tous les processeurs de telle façon que chaque processeur dispose d'un sous ensemble de tuples d'une table particulière. Le choix de telle ou telle stratégie de partitionnement de données a une influence considérable sur les performances et la complexité des algorithmes de Data Mining construits.

Assumant un partitionnement horizontal de données et une architecture à mémoire distribuée, plusieurs stratégies de distribution de données peuvent être considérées à savoir, la stratégie de distribution Round Robin, la stratégie de distribution en bloc, la stratégie de distribution aléatoire, et bien d'autres [Tan 2008] [Fre 1998].

4. 5. 3. 1 Stratégie de distribution Round Robin

Dans la stratégie Round Robin, les tuples de la base de données sont assignées aux différents processeurs de l'architecture parallèle d'une façon cyclique. Comme elle montre la figure 4. 10, la première tuple est assignée au premier processeur, la deuxième tuple est assignée au deuxième processeur, et ainsi de suite. Une fois le dernier processeur est atteint, le processus de distribution des tuples démarre de nouveau au premier processeur. À la fin de processus de distribution de données, tous les processeurs auront le même nombre de tuples, sauf si le nombre de tuples de la base de données n'est pas divisible sur le nombre de processeurs.

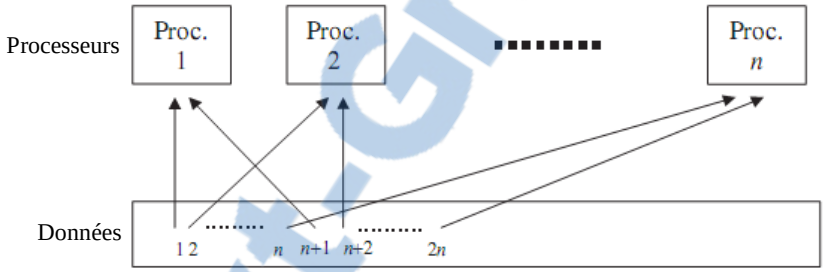


Figure 4. 10 Stratégie de distribution Round Robin

4. 5. 3. 2 Stratégie de distribution en blocs

Dans la stratégie de distribution en blocs, la base de données est partitionnée horizontalement en un certains nombre de partitions et chaque partition sera assignée à un processeur différent de l'architecture parallèle adoptée. Comme elle montre la figure 4. 11, la base de données est partition en n blocs, B_1, B_2, …, B_n, et chaque bloc B_i est assigné à un processeur *Proc* i.

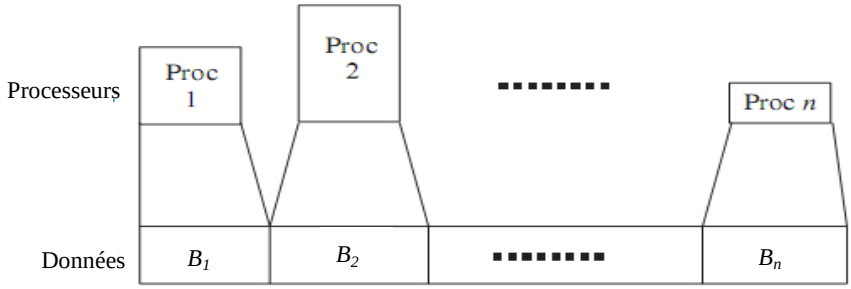


Figure 4. 11 Stratégie de distribution en blocs

4. 5. 3. 3 Stratégie de distribution aléatoire

Dans la stratégie de distribution aléatoire, une fonction aléatoire est appliquée afin de déterminer le processeur auquel une tuple de la base de données sera assignée. La figure ci-dessous illustre cette stratégie.

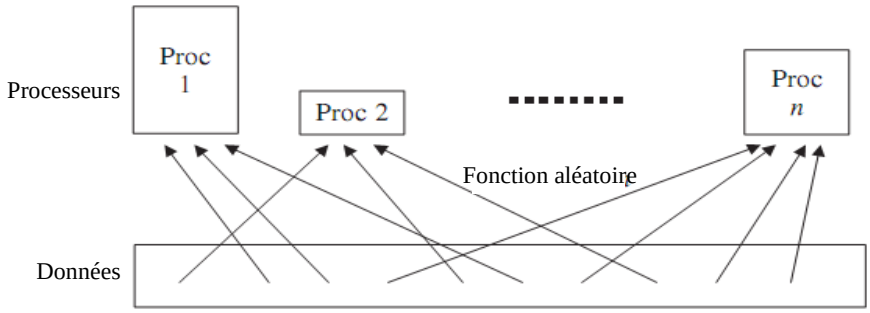


Figure 4. 12 Stratégie de distribution aléatoire

4. 5. 4 Les stratégies d'équilibrage de charge

L'équilibrage de charge couvre l'ensemble de techniques permettant de distribuer la charge de travail équitablement entre toutes les ressources de calcul d'un système parallèle et/ ou distribué. Afin d'optimiser les performances des algorithmes de Data Mining Parallèle et Distribué, il est primordial d'équilibrer la charge de travail à travers tous les ressources de calcul. En effet, une bonne technique d'équilibrage de charge améliore considérablement le temps de réponse de ces algorithmes, puisqu'elle permet une meilleure exploitation des ressources de calcul tout en réduisant le coût de communications. Deux approches d'équilibrage de charge sont adoptées en Data Mining Parallèle et Distribué [Zak 2000]: l'équilibrage de charge statique et l'équilibrage de charge dynamique.

4. 5. 4. 1 Équilibrage de charge statique

Dans l'équilibrage de charge statique, le partitionnement de la charge de travail et l'assignation des tâches aux ressources de calcul est effectuées avant l'exécution de l'algorithme de Data Mining Parallèle et Distribué ; aucune migration de tâches et/ ou déplacement de données n'est envisagé durant l'exécution afin de corriger la charge déséquilibrée. Les informations concernant les coûts des tâches et les caractéristiques dynamiques des ressources de calcul sont supposées connues a priori. Cette approche est efficace et simple à mettre en œuvre lorsque la charge de travail est au préalable suffisamment bien caractérisée. La plupart des algorithmes de Data Mining Parallèle et Distribué adaptent l'approche d'équilibrage de charge statique parce que ils assument des environnements dédiés et homogènes [Zak 2000].

4s. 5. 4. 2 Équilibrage de charge dynamique

Dans l'équilibrage de charge dynamique, l'assignation des tâches aux ressources de calcul se décide durant la phase d'exécution, en fonction des informations qui sont collectées sur l'état de charge du système. Ainsi, l'équilibrage de charge dynamique permet de redistribuer la charge de travail dynamiquement entre les ressources de calcul, en transférant les tâches d'une ressource surcharger vers une ressource sous-charger. La migration de tâches nécessite également un déplacement de données, puisque la ressource de calcul responsable d'une tâche à besoin également des données liées à cette tâche. Ainsi, l'équilibrage de charge dynamique engendre des coûts additionnels pour le déplacement des tâches/ données, mais il est intéressant lorsque la charge de travail est trop déséquilibrée ou trop dynamique. En effet, dans l'approche d'équilibrage de charge dynamique, pour arriver à un comportement efficace de système, il faut prendre en considération les coûts supplémentaires engendrés par cette approche (le coût de déplacement des tâches/ données, le coût de recueil des informations sur l'état de charge du système, etc.).

4. 6 Conclusion

Ce chapitre récapitule les différents problèmes liés à l'application et à la conception des techniques traditionnelles d'extraction de connaissances (ou de Data Mining) sur des environnements de calcul parallèle et distribué. Nous avons vus les différents problèmes rencontrés lors de parallélisation et de distribution des techniques traditionnelles de Data Mining. Par la suite, nous avons présentés les différents composants algorithmiques de conception parallèle et distribuée des algorithmes de Data Mining.

Quant au chapitre suivant, nous exposons les différentes approches et stratégies adoptées pour paralléliser et distribuer les techniques traditionnelles d'extraction de connaissances. L'objectif de ce chapitre est de présenter un état de l'art sur les différentes techniques parallèles et distribuées d'extraction de connaissances proposées dans la littérature.

CHAPITRE 5

Techniques Parallèles et Distribuées d'Extraction de Connaissances

5. 1 Introduction

Les techniques parallèles et distribuées d'extraction de connaissances (ou de Data Mining) proposées jusqu'à présent ne sont que des versions parallèles et distribuées pour des algorithmes séquentiels de Data Mining existants. La plupart de ces techniques ont leurs fondations sur des machines parallèles et elles sont ensuite appliquées sur des scénarios distribués. Les algorithmes de génération de règles d'association, plus précisément l'algorithme Apriori, sont les premiers des algorithmes de Data Mining qui ont subi le parallélisme. En effet, plusieurs approches de parallélisation et de distribution de l'algorithme Apriori ont été proposées dans la littérature à savoir, Count Distribution [Agr 1996], Data Distribution [Agr 1996], Hybrid Distribution [Han 1997], etc. Ensuite, plusieurs tentatives pour paralléliser et distribuer les autres algorithmes de Data Mining ont été faites ; Han et al. ont proposés plusieurs approches parallèle et distribuée de construction des arbres de décision [Han 1996], Stoffle et al. ont proposés une approche parallèle de l'algorithme *k*-means [Sto 1999], etc.

Dans ce chapitre, nous présentons pour chaque technique traditionnelle de Data Mining, présentée dans le chapitre 3, ces différentes approches parallèles et distribuées. Dans la section 5. 2, nous présentons les différentes approches de parallélisation et de distribution des algorithmes traditionnels de classification (*k*-Plus Proches Voisins, arbres de décision, réseaux bayésiens, réseaux de neurones artificiels). Ensuite, dans la section 5. 3, nous présentons les différentes approches de parallélisation et de distribution des algorithmes traditionnels de Clustering (*k*-means, Classification Hiérarchique Ascendante). Enfin, les différentes approches de parallélisation et de distribution des algorithmes de recherche de règles d'association, plus précisément de l'algorithme Apriori, sont présentées dans la section 5. 4.

5. 2 Classification parallèle et distribuée

La classification Parallèle et distribuée réfère à l'application des algorithmes traditionnels de classification sur un environnement de calcul parallèle et distribué, dont les données sont réparties sur les différents sites de cet environnement. Dans cette section, nous présentons les différentes approches de parallélisation et de distribution des algorithmes traditionnels de classification, tels que la méthode des *k*-Plus Proches Voisins, les arbres de décision, les réseaux bayésiens et les réseaux de neurones artificiels.

5. 2. 1 Parallélisation et distribution de la méthode *k*-Plus Proches Voisins

La méthode des *k*-Plus Proches Voisins est une méthode de raisonnement à partir de cas ; la phase d'apprentissage consiste simplement à stocker l'ensemble d'apprentissage et l'extraction de l'information se fait durant la phase de classification. Comme il est mentionné dans la section 3. 2. 2, la classification de chaque nouvel exemple x s'effectue selon deux étapes : la première étape consiste à calculer les distances entre x et tous les enregistrements de l'ensemble d'apprentissage selon une fonction de distance et ensuite, dans la deuxième étape, une fonction de combinaison des classes est appliquée pour calculer la classe de x en fonction des classes de ces *k* plus proches voisins.

Supposons que l'ensemble d'apprentissage est distribué entre les P unités de traitement d'un environnement de calcul parallèle et/ ou distribué. Dans ce cas, la classification de chaque nouvel exemple x en utilisant la méthode des *k*-Plus Proches Voisins peut être effectuée comme suit [Jin 2001] :

1. Sur chaque unité de traitement, calculer les *k* plus proches voisins locaux de x, en calculant les distances entre x et tous les enregistrements de l'ensemble d'apprentissage assigné à cette unité de traitement, en utilisant une fonction de distance.
2. Échanger les valeurs des distances et les classes des *k* plus proches voisins locaux entre les unités de traitement en utilisant une opération de réduction globale [Kum 1994] (voir la figure 5. 1).
3. Sur chaque unité de traitement, calculer les *k* plus proches voisins globaux de x à partir des *k* plus proches voisins locaux et calculer ensuite la classe de x en fonction des classes des ces *k* plus proches voisins globaux, en utilisant une fonction de combinaison des classes.

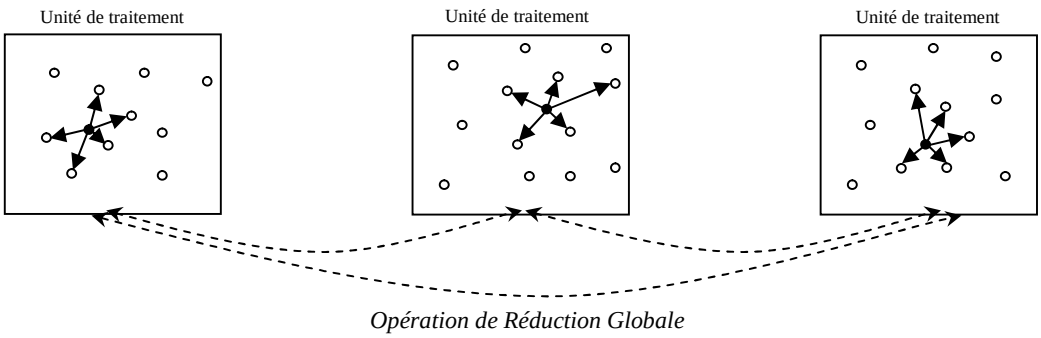


Figure 5. 1 Méthode *k*-Plus Proches Voisins parallèle et distribuée

5. 2. 2 Parallélisation et distribution des arbres de décision

Dans cette section, nous présentons les différentes approches de construction parallèle et distribuée des arbres de décisions. Ces approches sont en nombre de trois [Han 1996] [Sri 1997] [Sri 1999]: l'approche synchrone de construction de l'arbre de décision, l'approche partitionnée de construction de l'arbre de décision et l'approche hybride qui combine les deux approches précédentes. Dans ces trois approches, nous assumons un ensemble d'apprentissage de N exemples partitionnés d'une façon aléatoire entre les P unités de traitement de l'environnement de calcul parallèle et/ ou distribué, tel que à chaque unité de traitement est assigné N/P exemples d'apprentissages. Également, nous assumons que chaque exemple d'apprentissage contient A attributs à valeurs discrètes.

5. 2. 2. 1 Approche synchrone de construction de l'arbre

Dans cette approche, toutes les unités de traitement construisent d'une façon synchrone le même arbre de décision par l'envoi et la réception des informations de distribution des données locales (les valeurs des gains d'informations associés aux attributs des données locales). Les étapes principales de cette approche peuvent être résumées comme suit [Han 1996] :

1. Sur chaque unité de traitement, sélectionner un nœud à discriminer selon une stratégie de développement de l'arbre de décision (par exemple, la stratégie profondeur en premier ou la stratégie largeur en premier) et nommer ce nœud comme le nœud courant. Au début, le nœud racine est sélectionné comme le nœud courant.
2. Sur chaque unité de traitement, calculer la valeur de gain d'information local associée à chaque attribut du nœud courant.
3. Échanger les valeurs des gains d'informations locaux entre les unités de traitement en utilisant une opération de réduction globale [Kum 1994] (voir la figure 5. 2).

4. Sur chaque unité de traitement, calculer la valeur de gain d'information global associée à chaque attribut et sélectionner l'attribut qui discrimine le mieux le nœud courant.
5. Sur chaque unité de traitement, partitionner l'ensemble local d'exemples d'apprentissage entre les nœuds fils selon la satisfaction des tests de séparation sur l'attribut sélectionné.
6. Répéter les étapes 1-5 jusqu'à la satisfaction des conditions d'arrêts spécifiées.

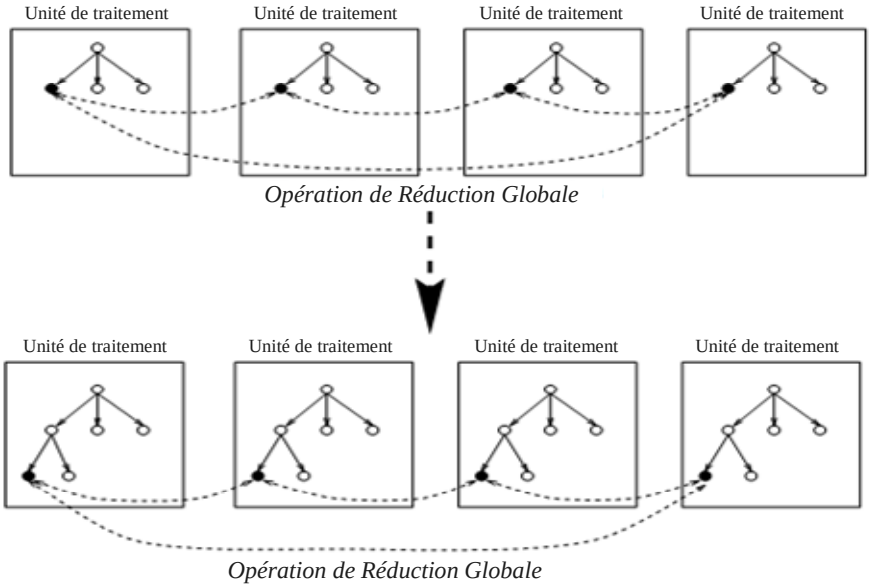


Figure 5. 2 Approche synchrone de construction de l'arbre de décision

5. 2. 2. 2 Approche partitionnée de construction de l'arbre

Dans cette approche, les unités de traitement coopèrent pour la construction de l'arbre de décision de telle sorte que chaque unité de traitement ou chaque groupe d'unités de traitement se focalise sur la construction des parties différentes de l'arbre de décision. En particulier, si plus d'une unité de traitement coopèrent pour développer un nœud de l'arbre de décision, alors ces unités de traitement sont distribuées pour développer les nœuds fils de ce nœud [Sri 1999]. Considérons le cas dans lequel un groupe d'unités de traitement P_n coopèrent pour développer un nœud n de l'arbre de décision. Le principe de cette approche est illustré sur la figure 5. 3 ; il peut être résumé dans les deux étapes suivantes :

Étape 1 : Les unités de traitement de groupe P_n coopèrent pour développer le nœud n selon le même principe que l'approche synchrone de construction de l'arbre de décision, décrite dans la section précédente.

Étape 2 : Une fois le nœud n est discriminé en k nœuds files, n_1, n_2, …, n_k, partitionner également le groupe P_n en plusieurs sous groupes d'unités de traitement et affecter à chaque sous groupe à un ou plusieurs nœuds fils. L'affectation des nœuds fils aux sous groupes de P_n s'effectue comme suit :

Cas 1 : Si le nombre de nœuds fils descendants de nœud n est plus grand que $|P_n|$, où $|P_n|$ est le nombre d'unités de traitement dans le groupe P_n.

1. Partitionner les nouds fils de nœud n en $|P_n|$ groupes de telle façon que le nombre total d'exemples d'apprentissage assignés à chaque groupe de nœuds est presque égale. Assigné chaque groupe de nœuds à une unité de traitement.
2. Échanger les exemples d'apprentissage de telle façon que chaque unité de traitement dispose des exemples qui contiennent tous les attributs qui appartiennent aux nœuds assignés à cette unité de traitement.
3. Maintenant, sur chaque unité de traitement, construire des sous arbres de décision d'une façon complètement indépendante en appliquant un algorithme séquentiel de construction de l'arbre de décision, tel que chaque sous arbre a comme racine l'un des nœuds de sous groupe de nœuds assigné à cette unité de traitement.

Cas 2 : Sinon (Si le nombre de nœuds fils descendants de nœud n est plus petit que $|P_n|$).

1. Affecter chaque nœud fils à un sous groupe de P_n de telle façon que le nombre d'unités de traitement assignés à chaque nœud est proportionnel au nombre d'exemples d'apprentissage correspondant à ce nœud.
2. Échanger les exemples d'apprentissage de telle façon que chaque groupe d'unité de traitement dispose des exemples d'apprentissage qui contiennent tous les attributs qui appartiennent au nœud assigné à ce groupe.
3. Les groupes d'unités de traitement assignés aux différents nœuds développe des sous arbres de décision d'une façon indépendante, tel que chaque groupe développe un sous arbre de racine le nœud assigné à ce groupe. Les groupes qui contiennent une unité de traitement

appliquent un algorithme séquentiel pour construire le sous arbre de racine le nœud assigné à cette unité de traitement. Les groupes qui contiennent plus d'une unité de traitement procèdent récursivement en répétant les étapes ci-dessus.

Au début, toutes les unités de traitement de l'environnement de calcul parallèle et/ ou distribué coopèrent pour développer le nœud racine de l'arbre de décision. À la fin de traitement, l'arbre complet est construit par la combinaison des sous arbres de chaque unité de traitement.

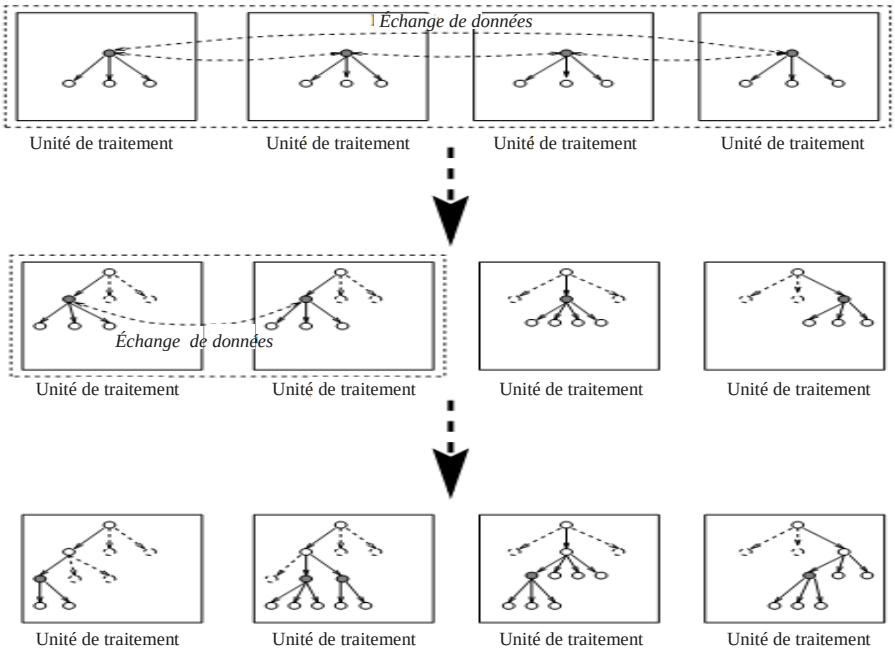
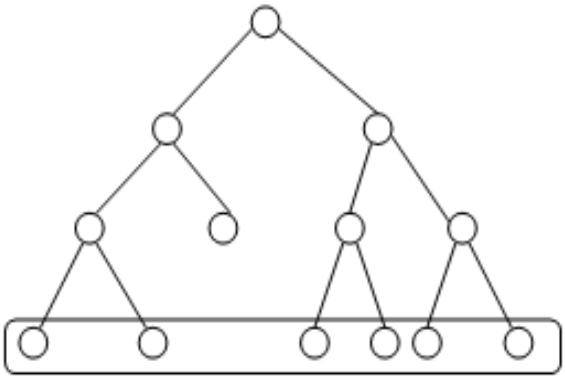


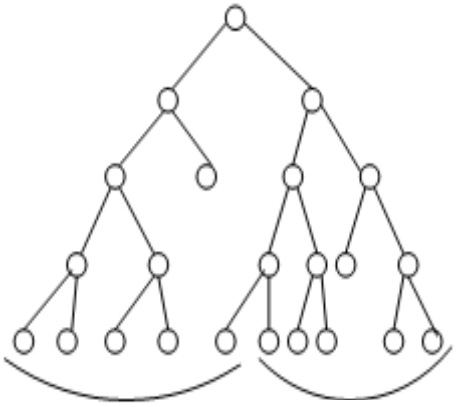
Figure 5. 3 Approche partitionnée de construction de l'arbre de décision

5. 2. 2. 3 Approche hybride

L'approche hybride combine les deux approches précédentes pour bénéficier de leurs bons paramètres. L'approche synchrone de construction de l'arbre de décision engendre un coût de communication élevé au l'augmentation de la frontière de l'arbre de décision. L'approche partitionnée de construction de l'arbre de décision engendre un coût de déplacement de données et d'équilibrage de charge élevé après chaque étape. Donc, pour remédier aux inconvénients de ces deux approches précédentes, l'approche hybride applique la première approche tant que le coût de communication n'est pas trop élevé. Une fois le coût de communication devient trop élevé, les unités de traitement et la frontière courante de l'arbre de décision sont partitionnés en deux parties [Sri 1999] ; ici, la deuxième approche est appliquée.

La figure 5. 4 illustre un exemple de construction de l'arbre de décision avec l'approche hybride. Jusqu'à au profondeur 3, les unités de traitement coopèrent selon l'approche synchrone pour développer l'arbre de décision (voir figure 5. 4 (a)), et au profondeur 4, l'approche partitionnée est appliquée, et la frontière courant de l'arbre de décision et les unités de traitement de l'environnement de calcul parallèle et/ ou distribué sont partitionnées en deux pour réduire le coût de communication (voir figure 5. 4 (b)).





(a) Approche synchrone de construction de l'arbre (b) Approche partitionnée de construction de l'arbre

Figure 5. 4 Approche hybride de construction de l'arbre de décision

L'élément clé de l'approche hybride est le choix de critère de partitionnement de l'arbre de décision. Si le partitionnement de l'arbre se fait fréquemment, alors l'approche hybride rapproche de l'approche partitionnée de construction de l'arbre de décision, et ainsi il engendre un coût de déplacement de données et d'équilibrage de charge trop élevé. Si le partitionnement de l'arbre ne se fait pas fréquemment, alors l'approche hybride souffre des coûts de communication élevés, comme l'approche synchrone de construction de l'arbre de décision. Une possibilité est de partitionner l'arbre de décision lorsque le coût de communication accumulé devient égal au coût de déplacement de données autour de la phase de partitionnement. Plus précisément, le partitionnement se fait lorsque [Han 1996] [Sri 1999] :

$$\sum \textit{Coût Communication} \geq \textit{Coût Déplacement} + \textit{Équilibrage Charge}$$

5. 2. 3 Parallélisation et distribution des réseaux bayésiens

Les réseaux bayésiens sont des modèles probabilistes graphiques qui représentent des connaissances incertaines [Pea 1988]. Graphiquement, un réseau bayésien est un graphe orienté sans circuit, dont à chaque nœud est associée une table de probabilités conditionnelles sur laquelle le modèle repose pour prédire des situations inconnues (voir la section 3. 2. 2).

L'apprentissage des réseaux bayésiens nécessite l'apprentissage de la structure du réseau (le graphe orienté) et les probabilités conditionnelles (les paramètres) associées au réseau. Dans [Ken 1997], une étude de différentes stratégies de distribution de la phase d'apprentissage des réseaux bayésiens (la structure et les paramètres) a été réalisée.

5. 2. 4 Parallélisation et distribution des réseaux de neurones artificiels

Dû à leurs structures modulaires, les réseaux de neurones artificiels peuvent être décomposés en plusieurs composantes exécutables en parallèles. En effet, en terme de granularité de décomposition de réseau, plusieurs stratégies de parallélisation et/ ou de distribution ont été considérées pour les réseaux de neurones artificiels [Won 1993] [Rog 1997a] [Rog 1997b] à savoir, la stratégie parallélisme entre exemples d'apprentissage, la stratégie parallélisme de blocs et la stratégie parallélisme de neurones.

5. 2. 4. 1 Parallélisme entre exemples d'apprentissage

Dans la stratégie parallélisme entre exemples d'apprentissage, chaque unité de traitement de l'environnement de calcul parallèle et/ ou distribué manipule le même réseau de neurones artificiels sur son ensemble local d'exemples d'apprentissage. Donc, cette stratégie n'exploite pas le parallélisme existant dans les réseaux de neurones lui-même ; ici, le parallélisme a eu lieu pour optimiser la phase d'apprentissage de réseau en distribuant l'ensemble d'apprentissage sur les différentes unités de traitement. La figure 5. 5 illustre les étapes d'apprentissage parallèle et/ ou distribué des réseaux de neurones artificiels en utilisant la stratégie parallélisme entre exemples d'apprentissage [Rog 1997b]. Ces étapes peuvent être résumées comme suit :

1. *Distribuer l'ensemble d'apprentissage* : Partitionner l'ensemble d'apprentissage en P partitions de taille égale, où P est le nombre d'unités de traitement de l'environnement de calcul parallèle et/ ou distribué. Ainsi, à chaque unité de traitement est assignée une partition de $1/P$ exemples d'apprentissage et manipule le même réseau de neurones artificiel sur cette partition.
2. *Calculer l'erreur locale de gradient* : Chaque unité de traitement calcul d'une façon itérative l'erreur locale de gradient sur sa partition locale d'exemples d'apprentissage.
3. *Diffuser les estimations locales de gradients* : Une fois une unité de traitement a complétée ces calculs locaux, elle diffuse les estimations locales de gradient à toutes les autres unités de traitement de l'environnement de calcul parallèle et/ ou distribué.
4. *Mettre à jour les poids du réseau* : Chaque unité de traitement mis à jour les poids de son réseau, après la réception des estimations locales de gradient calculées par les autres unités de traitement. Ainsi, à la fin de chaque itération, chaque unité de traitement aura

une copie identique de réseau de neurones avec des poids qui ont été estimés sur la totalité de l'ensemble d'apprentissage.

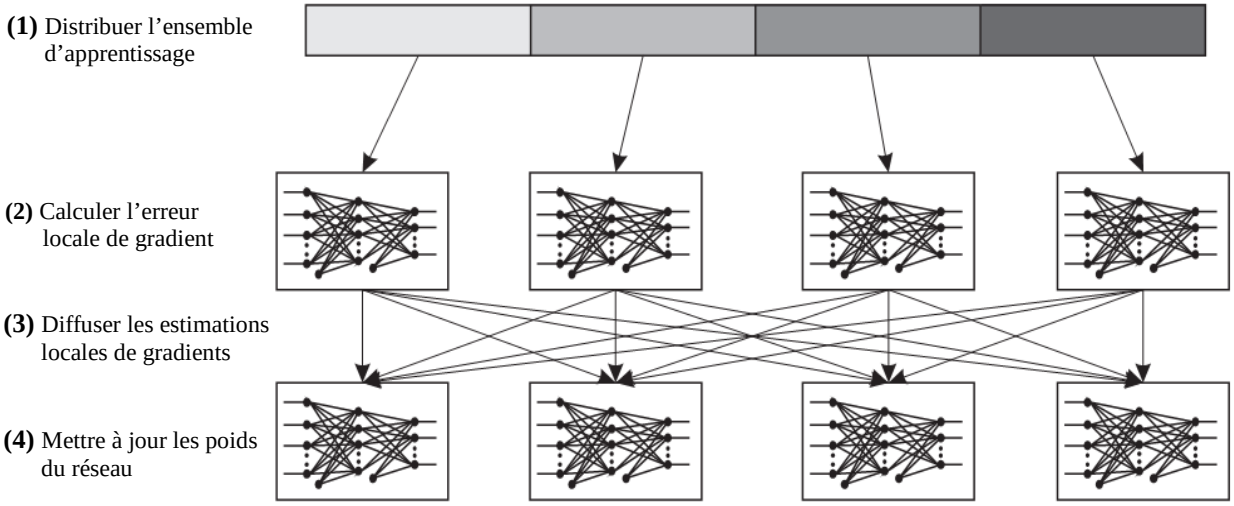


Figure 5. 5 Diagramme des étapes de la stratégie parallélisme entre exemples d'apprentissage

5. 2. 4. 2 Parallélisme de blocs

La stratégie parallélisme de blocs exploite le parallélisme existant dans les réseaux de neurones artificiels pour accélérer la phase d'apprentissage de réseau. En effet, dans cette stratégie, le réseau de neurones est partitionné en plusieurs blocs de neurones adjacents, où chaque bloc de neurones a une forme quasi rectangulaire avec une largeur x et une profondeur y (ou encore, chaque bloc de neurones contient x couches de y neurones). La figure 5. 6 illustre la stratégie parallélisme de blocs ; le partitionnement en blocs se base sur le parallélisme de couches existant dans les réseaux de neurones artificiels, où chaque bloc de neurones a une largeur $x = 1$ et une profondeur $y = M$ [Rog 1997b]. Chaque bloc de neurones est assigné à une unité de traitement différente de l'environnement de calcul parallèle et/ ou distribué.

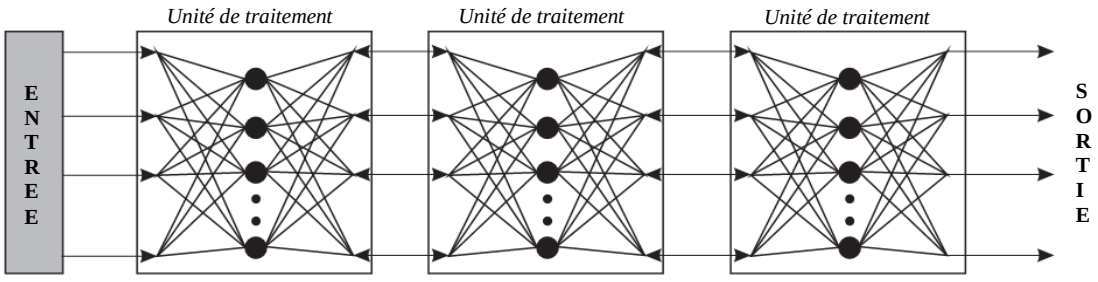


Figure 5. 6 Stratégie parallélisme de blocs ; partitionnement en blocs basé sur le parallélisme de couches (Largeur x =1 et Profondeur y =M)

Les unités de traitement contenant des neurones de la couche d'entrée débutent le processus d'apprentissage sur les données d'entrées. Leurs résultats de calcul seront propagés aux unités de traitement contenant des neurones des couches en avale. Une fois les unités de traitement contenant des neurones de la couche de sortie sont atteintes, le flux de calcul sera inversé, en propageant l'erreur signalée aux unités de traitement contenant des neurones des couches en amont.

5. 2. 4. 3 Parallélisme de neurones

L'unité de base d'un réseau de neurones artificiel est le neurone formel. Chaque neurone formel fonctionne indépendamment des autres neurones ; il traite les entrées qu'il reçoit, il ajuste ces poids et enfin il propage sa sortie calculée aux autres neurones. Donc, le neurone formel est un simple niveau de parallélisation des réseaux de neurones artificiels [Rog 1997b]. Dans la stratégie parallélisme de neurone, chaque neurone formel est traité comme un processus parallèle indépendant. À chaque unité de traitement de l'environnement de calcul parallèle et/ ou distribué est assigné un ou plusieurs neurones formels (ou processus parallèles) de telle façon que les neurones de chaque unité de traitement sont géométriquement indépendants. La figure 5. 7 illustre la stratégie parallélisme de neurones, où le partitionnement de réseau est effectué selon le parallélisme de neurones et chaque neurone est assigné à une unité de traitement différente.

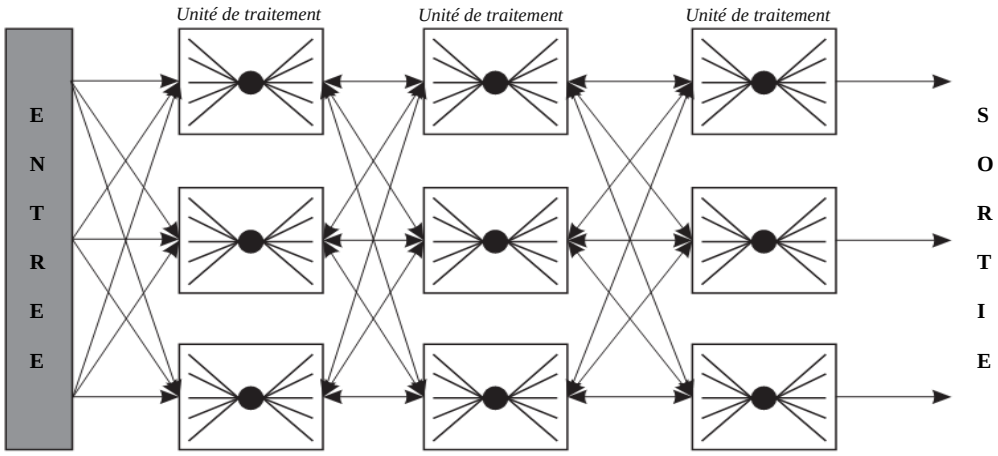


Figure 5. 7 Stratégie parallélisme de neurones

Le flux de calcul est similaire à celui de la stratégie parallélisme de blocs : Les unités de traitement contenant des neurones de la couche d'entrée débutent le processus d'apprentissage sur les données d'entrées. Les résultats de calcul des neurones de la couche d'entrée seront propagés aux unités de traitement contenant des neurones des couches en avale. Une fois les unités de traitement contenant des neurones de la couche de sortie sont atteintes, le flux de

calcul sera inversé, en propageant l'erreur signalée aux unités de traitement contenant des neurones des couches en amont.

5. 3 Clustering parallèle et distribué

Les algorithmes traditionnels de Clustering requièrent des données stockées sur une seule source de données centralisée. Actuellement, les applications engendrent des grandes quantités de données, généralement hétérogènes et éparpillées sur plusieurs sources de données distantes. La segmentation de telle catégorie de données nécessite le recours à des algorithmes parallèle et distribués de Clustering. Dans cette section, nous présentons les différentes approches de parallélisation et de distribution des algorithmes traditionnels de Clustering, tels que *k*-means et la Classification Hiérarchique Ascendante.

5. 3. 1 Parallélisation et distribution de la méthode *k*-means

La méthode *k*-means est la plus populaire des méthodes de Clustering par partitionnement. Il consiste à partitionner l'ensemble d'individus en k clusters par regroupement des individus autour de k centres choisis initialement au hasard. Le principe de fonctionnement de son algorithme séquentiel est bien détaillé dans la section 3. 3. 4, il peut être résumé comme suit : au début, l'algorithme *k*-means sélectionne aléatoirement k individus dans l'ensemble d'individus comme centres initiaux des clusters à construire et ensuite, basant sur une fonction de distance entre les individus et les centres des clusters, chaque individu de l'ensemble d'individus est affecté au cluster dont le centre est le plus proche de lui. Afin d'améliorer la qualité des clusters à construire, l'algorithme k-means procède récursivement en recalculant à chaque itération des nouveaux centres des clusters de telle façon que chaque cluster aura comme nouveau centre la moyenne de ces éléments. Ainsi, l'algorithme s'arrêt à la stabilité des clusters.

L'algorithme Parallèle *k*-means [Sto 1999] [Dhi 2000] exploite le parallélisme de données pour distribuer la charge de travail entre les processeurs de l'environnement de calcul parallèle et / ou distribué. La base de données est partitionnée en plusieurs partitions de taille égale, où chaque partition est assignée à un processeur. Puisque la tâche principale est de calculer et ainsi que de comparer les distances entre chaque objet et les centres des clusters, chaque processeur peut effectuer cette tâche d'une façon indépendante si les centres des clusters sont répliqués sur tous les processeurs. Les étapes de cet algorithme peuvent être résumées comme suit :

1. Partitionner l'ensemble d'individus en P partitions de taille N/P chacune, où P est le nombre de processeurs et N est le nombre d'individus dans l'ensemble d'individus. Ainsi, à chaque processeur est assignée une partition de l'ensemble d'individus.

2. Sélectionner aléatoirement k points m_j (j : 1, …, k) dans l'ensemble d'individus comme centres initiaux des clusters à construire et ensuite, assigné ces centres initiaux à tous les processeurs.
3. Sur chaque processeur, calculer d'une façon indépendante les clusters locaux en assignant chaque individu X_i de sa partition locale de l'ensemble d'individus au cluster C_j dont le centre m_j est le plus proche de X_i.
4. Sur chaque processeur, calculer les nouveaux centres des clusters en utilisant une opération de réduction globale [Kum 1994] (voir la figure 5. 8). En effet, durant cette opération, les processeurs sont synchronisés et chacun diffuse la somme et le nombre d'éléments de chaque cluster local aux autres processeurs.
5. Répété les étapes (3) à (5) jusqu'à la stabilité des clusters globaux, c'est-à-dire pas de changement dans les moyennes globaux.

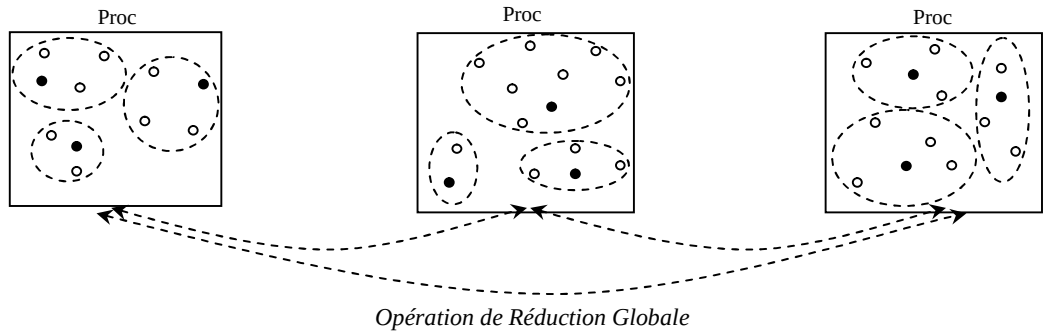


Figure 5. 8 Méthode *k*-means parallèle et distribuée

5. 3. 2 Parallélisation et distribution de Classification Hiérarchique Ascendante

La Classification Hiérarchique Ascendante permet de segmenter un ensemble d'individus en un certain nombre de segments (ou clusters) par regroupement récursif des clusters similaires. Débutant par des clusters contenant un seul individu, la Classification Hiérarchique Ascendante regroupe les clusters deux à deux, selon un critère d'agrégation des clusters, jusqu'à l'arrivé à un seul cluster contenant tous les individus de l'ensemble d'individus ou jusqu'à ce que une condition d'arrêt soit vérifiée (voir la section 3. 3. 4. 2). Les différents critères d'agrégation des groupements disjoints d'individus sont exposés dans la section 3. 3. 3.

La parallélisation et la distribution de la Classification Hiérarchique Ascendante est effectuée en assignant à chaque processeur un sous ensemble de clusters. Ainsi, l'agrégation des clusters s'effectue en parallèle, chaque processeur s'occupe de l'agrégation de quelques paires de clusters et chaque nouveau cluster construit sera assigné au processeur le moins

chargé. Les étapes de l'algorithme parallèle et distribué de Classification Hiérarchique Ascendante peuvent être résumées comme suit [Li 2006]:

1. Initialement, considérer chaque individu de l'ensemble d'individus comme un cluster et distribuer ces clusters d'une façon équitable entre les différents processeurs, où à chaque processeur est assigné N/P clusters, où P est le nombre de processeurs et N est le nombre d'individus dans l'ensemble d'individus.
2. Sur chaque processeur, calculer pour chaque cluster son cluster le plus proche en calculant la distance entre chaque paire de clusters. Les distances entre les paires de clusters sont maintiennent dans une matrice à deux dimension, distribué à travers tous les processeurs et chacun est responsable de ces colonnes correspondant à ses clusters assignés.
3. Tous les processeurs se synchronisent pour calculer les paires globales des clusters les plus proches. Ensuite, chaque paire de clusters sera regroupée pour construire un nouveau cluster de plus grande taille et il sera assigné au processeur le moins chargé.
4. Répéter les étapes (2) à (4) jusqu'à l'arrivée à un seul cluster contenant tous les individus de l'ensemble d'individus ou jusqu'à la satisfaction de certaines conditions d'arrêt.

5. 4 Règles d'association parallèles et distribuées

Les algorithmes parallèles et distribués de recherche de règles d'association se basent sur les algorithmes séquentiels existants, tel que DHP [Par 1995], Partition [Sav 1995] et DIC [Bri 1997], ainsi que l'algorithme Apriori qui constitue l'algorithme de base pour la majorité de ces algorithmes. Zaki a présenté dans [Zak 1999], une classification de ces algorithmes selon la stratégie d'équilibrage de charge, l'architecture et le type de parallélisme exploité. Dans cette section, Nous décrivons les différentes approches parallèles et distribuées de recherche de règles d'association et ensuite, afin d'expliquer le principe de chaque approche, nous détaillons quelques exemples d'algorithmes parallèles et distribués de recherche de règles d'association.

5. 4. 1 Parallélisation et distribution des règles d'association

Basant sur la façon dont les itemsets candidats sont distribués entre les différentes unités de traitement de l'environnement de calcul parallèle et/ ou distribué, trois approches principales ont été adoptées dans les algorithmes parallèles et distribués de recherche de règles d'association à savoir, l'approche réplication des itemsets candidats, l'approche partitionnement des itemsets candidats (sans réplication) et l'approche hybride qui assume une réplication partielle des itemsets candidats.

5. 4. 1. 1 Approche réplication des itemsets candidats

Dans cette approche, la base de données est distribuée d'une façon équitable entre les différentes unités de traitement de l'environnement de calcul parallèle et /ou distribué. Cependant, le processus de génération des itemsets candidats est répliqué sur toutes les unités de traitement. Ainsi, à chaque itération de l'algorithme, le même ensemble d'itemsets candidats est traité sur toutes les unités de traitement, d'où l'appellation « *réplication des itemsets candidats* ». Plusieurs algorithmes parallèles et distribués de recherche de règles d'association sont fondés cette approches, à savoir : Count Distribution (CD) [Agr 1996], Non-Partitioned Apriori (NPA) [Shi 1996], Parallel Data Mining (PDM) [Par 1995], Distributed Mining Algorithm (DMA) [Che 1996b] et Common Candidate Partitioned Database (CCPD) [Zak 1996]. Afin d'expliquer le principe de l'approche réplication des itemsets candidats, nous détaillons dans la section 5. 4. 2 l'algorithme Count Distribution.

5. 4. 1. 2 Approche partitionnement des itemsets candidats

Dans cette approche, en plus de partitionnement de la base de données, l'ensemble des itemsets candidats est partitionné entre les différentes unités de traitement de l'environnement de calcul parallèle et/ ou distribué. Ainsi, Cette approche adresse le problème de mémoire des algorithmes qui adoptent l'approche réplication des itemsets candidats par le partitionnement des itemsets candidats entre les différentes unités de traitement. Parmi les algorithmes qui adoptent l'approche partitionnement des itemsets candidats on trouve : Data Distribution (DD) [Agr 1996], Intelligent Data Distribution (IDD) [Han 1997], Simply-Partitioned Apriori (SPA) [Shi 1996], Hash-Partitioned Apriori (HPA) [Shi 1996] et Partitioned Candidate Common Database (PCCD) [Zak 1996]. Le principe de cette approche est expliqué dans la section 5. 4. 2, en détaillant l'algorithme Data Distribution et l'algorithme Intelligent Data Distribution.

5. 4. 1. 3 Approche hybride

L'approche hybride qui combine les deux approches précédentes est aussi possible, et peut être souhaitable pour exploiter toutes les disponibilités de parallélisme dans les méthodes de recherche de règles d'association. L'approche réplication des itemsets candidats assume une réplication totale des itemsets candidats et l'approche partitionnement des itemsets candidats assume un partitionnement sans réplication des itemsets candidats. Cependant, l'approche hybride assume une réplication partielle des itemsets candidats, d'où l'appellation l'« approche réplication partielle des itemsets candidats ». Parmi les algorithmes parallèles et distribués de recherche de règles d'association qui adoptent la réplication partielle des itemsets candidats on trouve : Hybrid Distribution (HD) [Han 1997], Candidate Distribution

[Agr 1996] et HPA-ELD [Shi 1996]. Le principe de cette approche est bien détaillé dans la section 5. 4. 2, on décrivant l'algorithme Hybrid Distribution.

5. 4. 2 Exemples d'algorithmes parallèles et distribués de recherche de règles d'association

Afin d'expliquer les différentes approches parallèles et distribuées de recherche de règles d'association (réplication totale, partitionnement sans réplication et réplication partielle des itemsets candidats), nous présentons dans cette section quelques exemples d'algorithmes parallèles et distribués de recherche de règles d'association.

5. 4. 2. 1 Count Distribution

Dans cette formulation parallèle de l'algorithme Apriori, proposé par Agrawal et Shafer en 1996 [Agr 1996], la base de données est partitionnée horizontalement en n partitions, $\{D^1, D^2, \ldots, D^n\}$, distribuées sur n processeurs, $\{P^1, P^2, \ldots, P^n\}$. Cependant, l'ensemble d'itemsets candidats C_k est répliqué sur tous les processeurs. A chaque itération de l'algorithme, chaque processeur P^i calcule les nombres d'apparitions des itemsets candidats C_k sur ses données locales D^i. Ainsi, puisque le nombre d'apparition de chaque itemset candidat sur chaque processeur est partielle, les processeurs se synchronise pour l'échange de leurs résultats de calcul locaux (les nombres d'apparition locaux des itemsets candidats) ; d'où le terme « *Count Distribution* ». En effet, le nombre d'apparition globale de chaque itemset candidat sur la totalité de la base de données est obtenu sur chaque processeur par la somme de tous ces nombres d'apparitions locaux en utilisant une opération de réduction globale [Kum 1994] (voir la figure 5. 9). Ainsi, chaque processeur peut identifier maintenant l'ensemble des itemsets fréquents L_k à partir de l'ensemble des itemsets candidats C_k ; Puisque l'ensemble des itemsets candidats C_k est identique sur tous les processeurs, alors l'ensemble des itemsets fréquents L_k est lui même identique sur tous les processeurs. On se basant sur cette connaissance globale L_k, l'ensemble d'itemsets candidats de taille $k+1$, C_{k+1}, pour la prochaine itération est généré sur chaque processeur. L'algorithme Count Distribution s'arrête lorsqu'il n'y a plus d'itemsets candidats a générés.

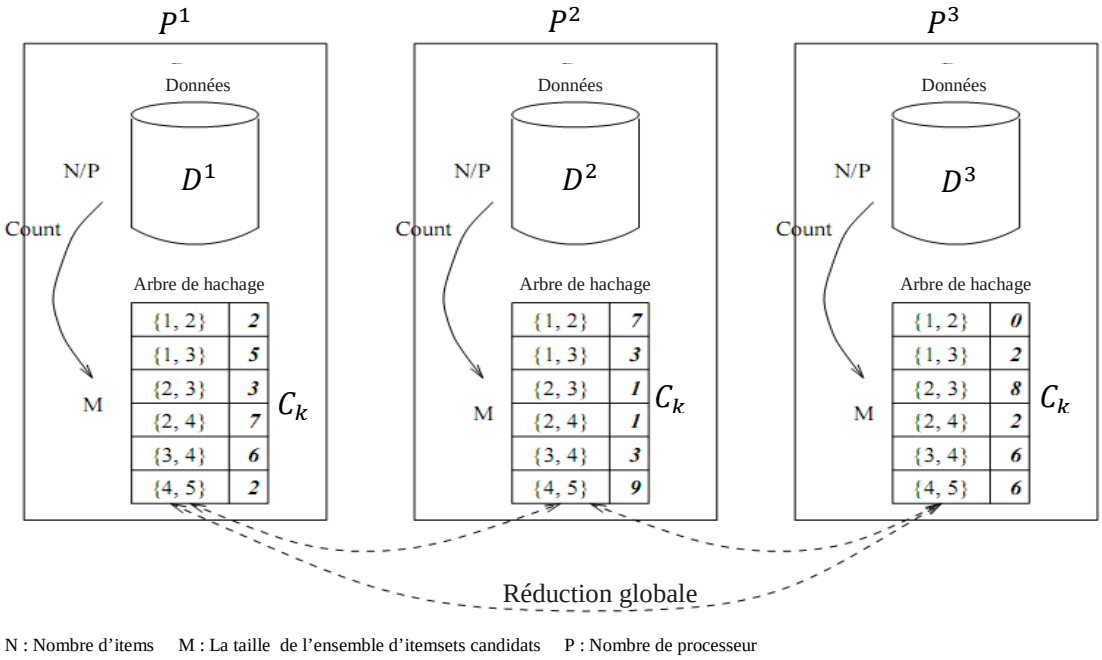


Figure 5. 9 Algorithme Count Distribution

5. 4. 2. 2 Data Distribution

Dans l'algorithme Data Distribution (DD), proposé par Agrawal et Shafer en 1996 [Agr 1996], en plus de partitionnement de la base de données, l'ensemble des itemsets candidats C_k est partitionné en n ensembles disjoints, $\{C^1, C^2, \ldots, C^n\}$, distribués sur les n processeurs, $\{P^1, P^2, \ldots, P^n\}$. Ce partitionnement se fait selon le mode Round Robin. En effet, chaque processeur P^i est responsable pour calculer les nombres d'apparitions des itemsets candidats de son ensemble local C^i sur la totalité de la base de données. Pour réaliser cette tâche, chaque processeur besoin de parcourir sa partition locale de données aussi bien que les partitions des autres processeurs. Donc, chaque processeurs broadcast sa partition locale à tous les autres processeurs et reçoit les partitions des autres processeurs, et ensuite il parcourt les partitions recevaient pour calculer les nombres d'apparitions totales de leurs candidats locaux sur la totalité de la base de données. Alors, chaque processeur détermine un sous ensemble d'itemsets fréquents L^i à partir de son ensemble local d'itemsets candidats C^i. Cet ensemble d'itemsets fréquents est ensuite transmis à tous les autres processeurs par une opération de diffusion générale (*all-to-all broadcast*). En conséquent, tous les processeurs ont maintenant l'ensemble L_k, tel que $L_k = \bigcup_{i=1}^{n} L_k^i$, et génèrent l'ensemble C_{k+1} qu'il sera partitionné sur tous les processeurs. Ce processus s'arrête lorsqu'il n'y a plus d'itemsets candidats a générés. Le principe de l'algorithme DD est illustré sur la figure ci-dessous.

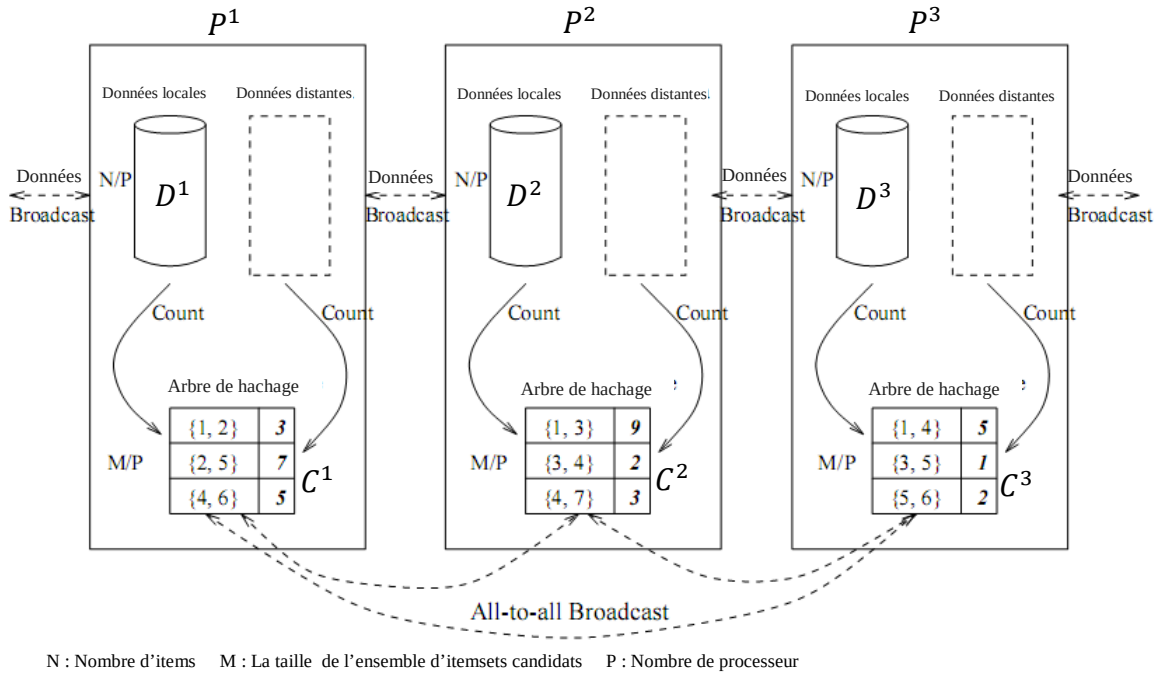


Figure 5. 10 Algorithme Data Distribution

5. 4. 2. 3 Intelligent Data Distribution

L'algorithme Intelligent Data Distribution (IDD), proposé par Han et al. en 1997 [Han 1997], est développé pour remédier au problème de temps de communication élevé de l'algorithme Data Distribution, provoqué par des opérations d'envoie et de réception des buffers de données assez volumineux. Dans l'algorithme IDD, les partitions locales sont envoyées à tous les autres processeurs par une opération de diffusion basée sur l'anneau (*ring-based all-to-all broadcast*). En effet, les processeurs sont vues comme étant un anneau logique, où chacun connaît son prédécesseur et son successeur et possède un buffer d'émission et un buffer de réception. Comparé à l'algorithme DD, où tous les processeurs diffusent leurs données à tous les autres processeurs, l'algorithme IDD exécute seulement une communication point-a-point entre les voisins, réduisant ainsi le nombre de communications.

Un autre problème de l'algorithme DD que IDD a amélioré est la redondance de calcul. Afin d'éliminer la redondance de calcul dû au partitionnement de l'ensemble des itemsets candidats, l'algorithme IDD trouve une méthode rapide pour vérifier si une transaction donnée peut contenir potentiellement tous les itemsets candidats stockés à chaque processeur. En effet, l'algorithme IDD n'adopte pas le partitionnement en mode Round Robin. Cependant, si l'ensemble des itemsets candidats C_k est partitionné entre les processeurs d'une telle façon que chaque processeurs obtient seulement les itemsets qui débute avec un sous ensemble de l'ensemble de tous les items possible, alors les items d'une transaction peuvent être vérifiés à cet sous ensemble pour déterminer si un arbre de hachage contient des

itemsets candidats débutant avec ces items. L'arbre de hachage est traversé seulement avec les items d'une transaction qui appartiennent à cet sous ensemble. Ainsi, le problème de redondance de calcul de l'algorithme DD est résolu par le partitionnement intelligent de C_k .

La figure 5. 11 illustre ce partitionnement intelligent, où tous les itemsets candidats de chaque processeur débutent par les items stockés dans son Bit-Map.

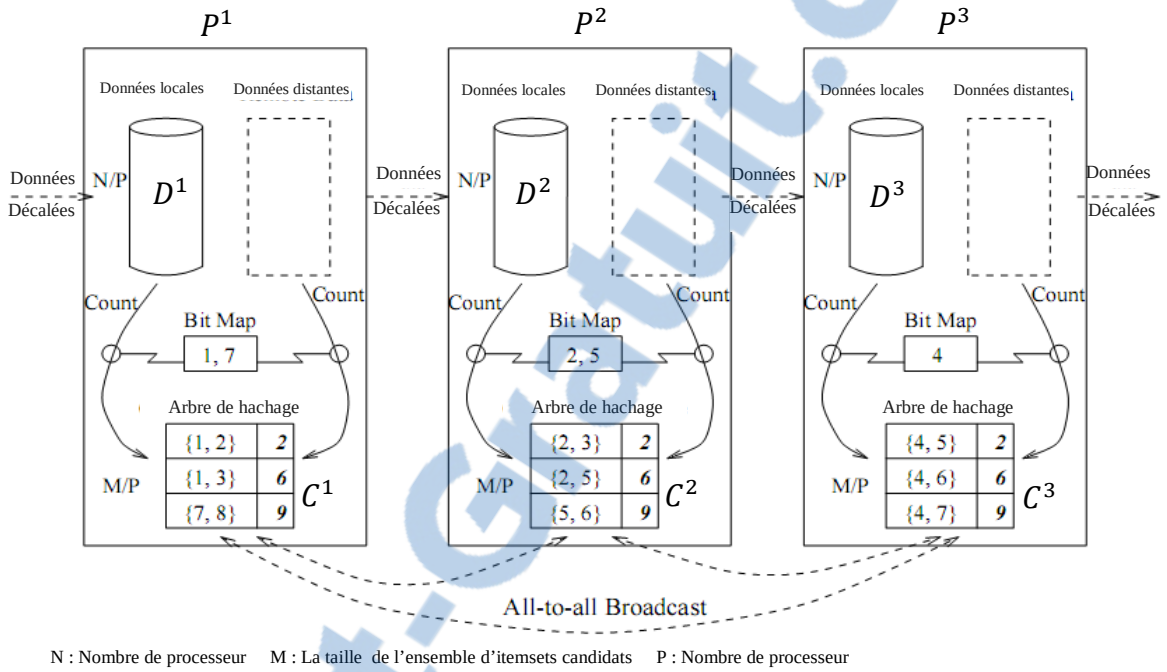


Figure 5. 11 Algorithme Intelligent Data Distribution

5. 4. 2. 4 Candidate Distribution

L'algorithme Candidate Distribution, proposé par Agrawal et Shafer en 1996 [Agr 1996], tente de réduire les overheads de communication et de synchronisation dans les deux algorithmes : Count Distribution et Data Distribution. En effet, l'algorithme Candidate Distribution partitionne la base de données et l'ensemble des itemsets candidats C_k de telle façon que chaque processeur peut procéder indépendamment des autres. Durant la $l^{ème}$ itération, où l est déterminé à base des heuristiques sur le support, l'algorithme partitionne l'ensemble des itemsets candidats C_l entre les processeurs de telle façon que chaque processeur P^i peut généré un ensemble d'itemsets candidats C_m^i $(m > l)$ indépendamment de tous les autres processeurs ($C_m^i \bigcap C_m^j = \phi$, $i \neq j \ \ \forall \ \ i = 1,...n \ \ \wedge \ \ j = 1,...,n$). Au même temps, la base de données est redistribuée de telle sorte que chaque processeur P^i peut calculer le support global de tous les itemsets candidats dans C_m^i indépendamment de tous les autres processeurs. Noté que, dépendant de la qualité de partitionnement des itemsets candidats, des

portions de la base de données peuvent être répliquées sur plusieurs processeurs. Le partitionnement de l'ensemble d'itemsets candidats C_l se fait par groupement d'itemsets candidats ayant $l-1$ préfixes communs. Par exemple, soit $\varepsilon = \{ABC, ABD, ABE\}$ un ensemble d'itemsets ayant le préfixe commun AB, la procédure de génération des itemsets candidats de l'algorithme Apriori génère ces itemsets $ABCD, ABCE, ABDE$ et $ABCDE$ par la jointure seulement des itemsets dans ε. Donc, dans l'algorithme Candidate Distribution, l'algorithme de partitionnement exploite la sémantique de la procédure de génération des itemsets candidats de l'algorithme Apriori.

Après la distribution des itemsets candidats, chaque processeur peut procéder indépendamment des autres ; il calcul le support de tous ces itemsets candidats sur sa partition de données. Pas d'échange de supports ni de données entre les processeurs, alors pas de communications ni synchronisations entre les processeurs pour calculer le support global de leurs itemsets candidats ni pour générer l'ensemble global d'itemsets candidats. La seule dépendance entre les processeurs est pour l'élagage de l'ensemble local d'itemsets candidats durant la phase d'élagage. Cependant, cette information est envoyée d'une façon asynchrone, et les processeurs ne sont pas suspendus en attente de l'arrivée de l'information de l'élagage à partir de tous les autres processeurs. Durant la phase d'élagage, le processeur simplifie l'ensemble local d'itemsets candidats autant que possible en utilisant l'information arrivée, et il débute la tâche de calcul de support des itemsets candidats. Cependant, les informations de l'élagage arrivées en retard peuvent être exploitées durant les prochaines itérations.

Noté que, avant le partitionnement de l'ensemble des itemsets candidats (c'est-à-dire avant la $l^{ème}$ itération), l'algorithme Candidate Distribution utilise soit l'algorithme Count Distribution soit l'algorithme Data Distribution.

5. 4. 2. 4 Hybrid Distribution

L'algorithme Hybrid Distribution, proposé par Han et al. [Han 1997], combine l'algorithme Count Distribution et l'algorithme Intelligent Data distribution afin de bénéficier de leurs avantages et de réduire ainsi leurs inconvénients. L'algorithme Hybrid Distribution assume l'existence de P processeurs partitionnés en G groupes, où chaque groupe contient P/G processeurs. Également, il assume que la base de données est partitionnée d'une façon équitable en P/G partitions de taille $\frac{N}{(P/G)}$ chacune, où N est le nombre de transactions dans la base de données, et chaque partition est assignée à un processeur de chaque groupe de processeurs ; ainsi la base de données est partitionnée entre les processeurs de chaque groupes et répliquée sur tous les groupes. De même, à chaque itération de l'algorithme, l'ensemble des itemsets candidats C_k est partitionné d'une façon équitable en G sous ensembles de taille

M/G chacun, où M est le nombre d'itemsets de l'ensemble C_k, et chaque sous ensemble d'itemsets candidats est assigné à un groupe de processeurs. L'algorithme Hybride distribution exécute l'algorithme Count distribution entre les processeurs de chaque groupe et l'algorithme Intelligent Data distribution entre les groupes.

5. 5 Conclusion

Pour conclure le chapitre, beaucoup de travaux ont été effectués afin de paralléliser et de distribuer les techniques traditionnelles d'extraction de connaissances (ou de Data Mining), mais peut de ces travaux qui s'intéressent à l'extraction de connaissances à partir des entrepôts de données, qui sont le plus souvent répartis sur des sites distants (éloignés géographiquement).

Dans le chapitre suivant, on s'intéresse à cette problématique en proposant une approche parallèle et distribuée d'extraction de connaissances exprimables sous forme de règles d'association à partir des entrepôts de données répartis.

CHAPITRE 6

Nouvelle Approche Parallèle et Distribuée d'Extraction de Connaissances

6. 1 Introduction

L'extraction de connaissances à partir des entrepôts de données distribués nécessite le recours à des approches parallèles et distribuées d'extraction de connaissances. Dans le domaine de l'EPDC, plusieurs solutions ont été proposées pour répondre au problème d'extraction de connaissances à partir des grands volumes de données distribués, mais très peu d'algorithmes proposés dans ce contexte qui s'intéressent à l'extraction de connaissances à partir des entrepôts de données distribués. Les connaissances à extraire à partir de ces derniers peuvent prendre plusieurs formes à savoir, les modèles mathématiques ou logiques, les règles de classification ou d'association, ou toutes autres formes compréhensibles qui peuvent être interprétées en connaissances utiles et intéressantes. Due à sa simplicité (facilement interprétables en connaissances) et la simplicité de ses algorithmes, les règles d'association constituent l'une des formes de connaissances les plus populaires de domaine de l'ECD. En effet, dés son introduction dans [Agr 1993], les algorithmes de recherche de règles d'association ont été employés avec beaucoup de succès dans divers secteurs, citant la conception de catalogues, la segmentation de la clientèle, les préventions financières, les politiques de marketing, les diagnostiques médicales, et bien d'autres ; aucun domaine d'application n'est a priori exclu.

Dans notre travail on s'intéresse plus précisément aux connaissances extraites sous forme de règles d'association, en proposant une approche parallèle et distribuée d'extraction de règles d'association à partir d'un entrepôt de données distribué. Avant d'expliquer le principe de cette approche (section 6. 8), nous décrivons, dans cet ordre, le problème d'extraction de règles d'association à partir des entrepôts de données distribués (section 6. 2), l'architecture et le configuration du réseau assumée par l'approche proposée (section 6. 3) et la stratégie de distribution de données assumée par cette approche (section 6. 4). Par la suite, nous détaillons le principe de l'algorithme proposé pour la génération des itemsets fréquents sur chaque site du réseau spécifié (section 6. 6), ainsi que le principe de l'algorithme proposé

pour la mise à jour de ces itemsets fréquents en cas de rafraîchissement de l'entrepôt de données (section 6. 7).

6. 2 Notations et formulation du problème

Soit $W = \{ W_1, W_2, \ldots, W_n \}$ un entrepôt de données distribué sur un ensemble de n sites, $S = \{ S_1, S_2, \ldots, S_n \}$, où à chaque site S_i est assigné un fragment W_i de l'entrepôt de données W. Soit $I = \{ I_1, I_2, \ldots, I_n \}$ l'ensemble des ensembles d'items (attributs ou objets) de tous les fragments $W_{i\ (i=\overline{1,n})}$ de l'entrepôt de données W, tel que $I_i = \{ I_i^1, I_i^2, \ldots, I_i^{m_i} \}$ est l'ensemble d'items du fragment W_i. Chaque fragment W_i de l'entrepôt de données W contient M_i transactions, où chaque transaction $t \in W_i$ est un sous ensemble de l'ensemble d'items I_i, $t \subset I_i$. À chaque transaction t est associé un identificateur unique, appelé TID (*Transaction IDentification*).

Comme elle montre la figure 6. 1, le problème d'extraction de toutes les règles d'association pertinentes à partir d'un entrepôt de données distribué peut être subdivisé en trois sous problèmes suivants :

1. Sur chaque site S_i, trouver l'ensemble de tous les itemsets $x \subset I_i$ dont le support dépasse un certain seuil (*minsup*) spécifié par les utilisateurs. Cet ensemble, noté L_i^L, est appelé l'ensemble des « *itemsets localement fréquents* » sur le fragment W_i de l'entrepôt de données W.

$$L_i^L = \{ x / \ x \subset I_i \ \wedge \ Support(x) = \frac{|x|}{M_i} \geq minsup \} \qquad \forall$$

$i \in \{1,...,n\}$

Où $|x|$ est le nombre de transactions de W_i comportant l'itemset x et M_i est le nombre total de transactions de W_i.

2. Une fois les ensembles L_i^L des itemsets localement fréquents sont identifiés sur chaque fragment W_i de chaque site S_i, une étape de fusion et de combinaison de ces ensembles a eu lieu afin de générer l'ensemble L^G des « *itemsets globalement fréquents* » sur l'entrepôt de données global, W.

$$L^G = fusion_{i=1}^{n} (L_i^L)$$

3. Utiliser l'ensemble L^G des itemsets globalement fréquents pour générer toutes les règles d'association dont la confiance dépasse un certain seuil (*minconf*) spécifié par les utilisateurs.

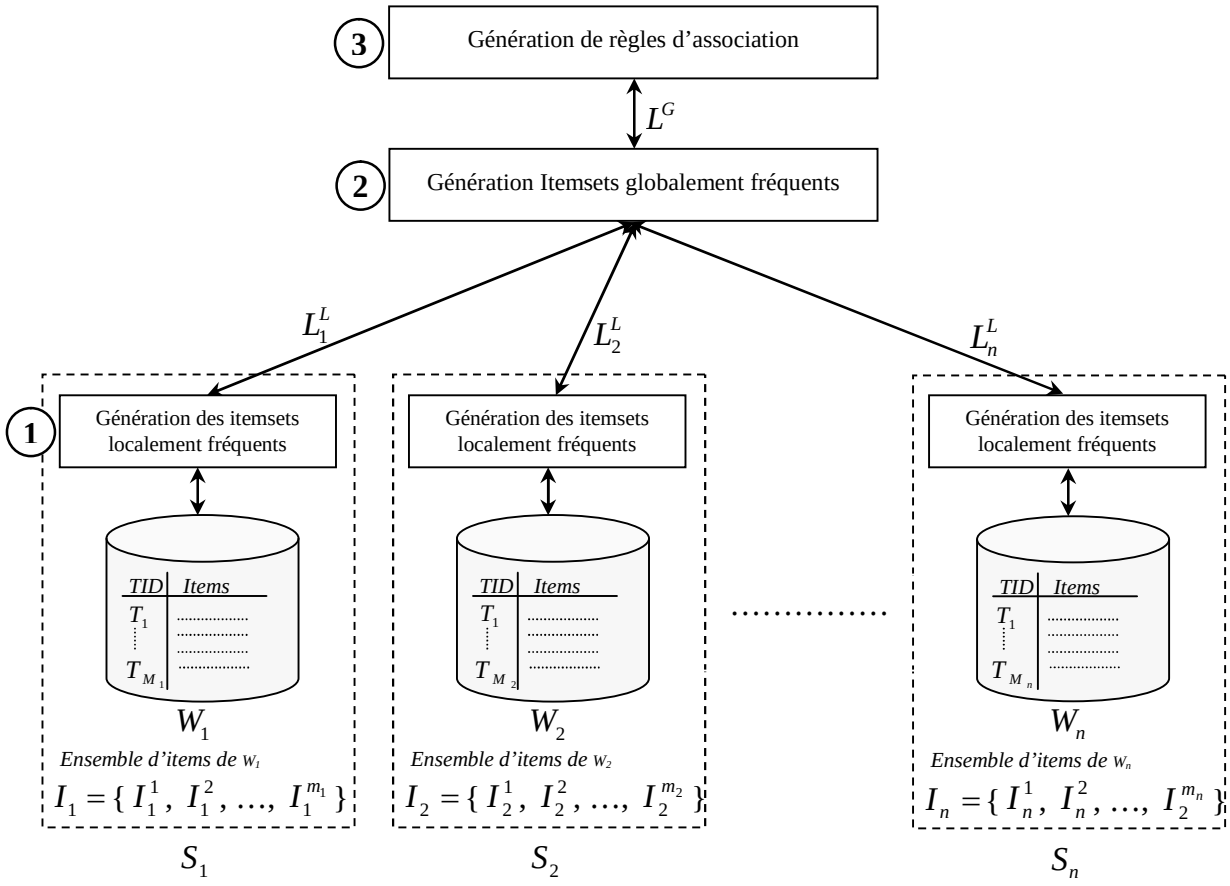


Figure 6.1 Extraction de règles d'association à partir d'un entrepôt de données distribué

6. 3 Architecture et configuration du réseau

Notre approche décrite dans la section 6. 8, assume l'architecture illustrée sur la figure 6. 3. On suppose que l'entrepôt de données manipulé est distribué à travers n sites, $S = \{ S_1, S_2, \ldots, S_n \}$, et sur chaque site S_i du réseau, le fragment W_i de l'entrepôt de données W est manipulé par un groupe de P processeurs à mémoire distribuée, $GP = \{ P_i^0, P_i^1, \ldots, P_i^{P-1} \}$; notre approche assume la disponibilité de au moins de deux processeurs sur chaque site du réseau, un est configuré comme maître et l'autre ou les autres sont des processeurs esclaves. Chaque processeur est identifié par un nombre unique, appelé ID (IDentificateur de processeur); 0 : pour le processeur maître et $1,...,P-1$: pour les processeurs esclaves. Le but d'utilisation d'une configuration Maître-Esclave au sein de chaque nœud du réseau est d'éviter les synchronisations coûteuses entre les processeurs. Ainsi, chaque processeur peut transférer d'une manière asynchrone ces résultats locaux au processeur maître qui s'occupe de leurs combinaisons en résultats globaux. En plus, notre

approche assume que les processeurs de chaque site S_i et son entrepôt W_i sont connectées entre eux à travers un réseau d'interconnections rapide (de haut débit). L'architecture d'un site S_i du réseau est illustrée sur la figure ci-dessous.

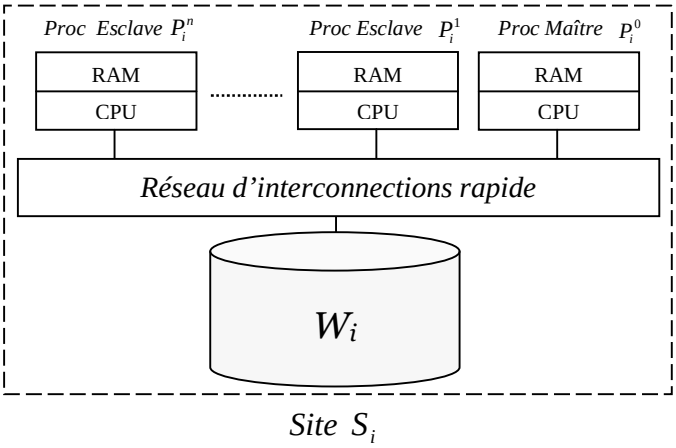


Figure 6. 2 Architecture d'un nœud *i* du réseau

Également, comme elle montre la figure 6. 3, tous les sites du réseau et le site superviseur sont connectés entre eux à travers un réseau d'interconnections lent. L'objectif de configuration d'un site superviseur est d'allouer la tâche de combinaison des résultats locaux en résultats globaux à un site central. En effet, le site superviseur se communique avec les processeurs maîtres de tous les sites S_i du réseau pour récupérer les résultats locaux.

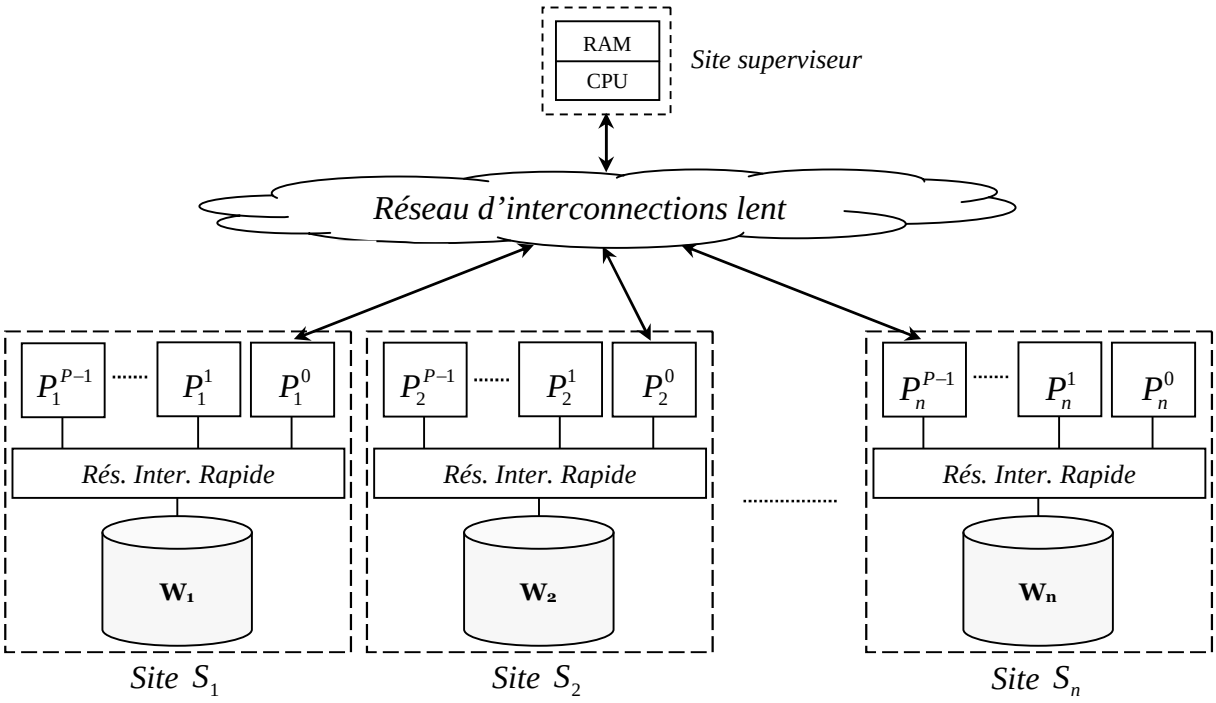


Figure 6. 3 Architecture et configuration du réseau dans l'approche proposée

6. 4 Distribution de données

La distribution de données indique la façon de répartition des données de l'entrepôt de données entre les différents processeurs de chaque site du réseau. Notre approche, décrite dans la section 6. 8, assume un partitionnement horizontal de données sur tous les sites du réseau. En effet, sur chaque site S_i, le fragment W_i de l'entrepôt de données W est partitionné horizontalement en P partitions disjointes de taille égale à M_i / P, où M_i est la taille de l'entrepôt W_i, et chaque partition de données W_i^j est assignée à un processeur P_i^j ; elle constitue l'espace de données manipulé par ce processeur. Comme elle montre la figure 6. 4, à chaque processeur P_i^j du site S_i est assigné un espace de données W_i^j de l'espace de données de W_i. Également, chaque espace de données W_i^j est partitionné horizontalement en un certain nombre de partitions disjointes, D_k, de taille peuvent être chargées en mémoire centrale. En conséquence, chaque processeur P_i^j du site S_i manipule son espace de données W_i^j partition par partition. Cette stratégie de distribution de données assumée sur chaque site du réseau est illustrée sur la figure ci-dessous.

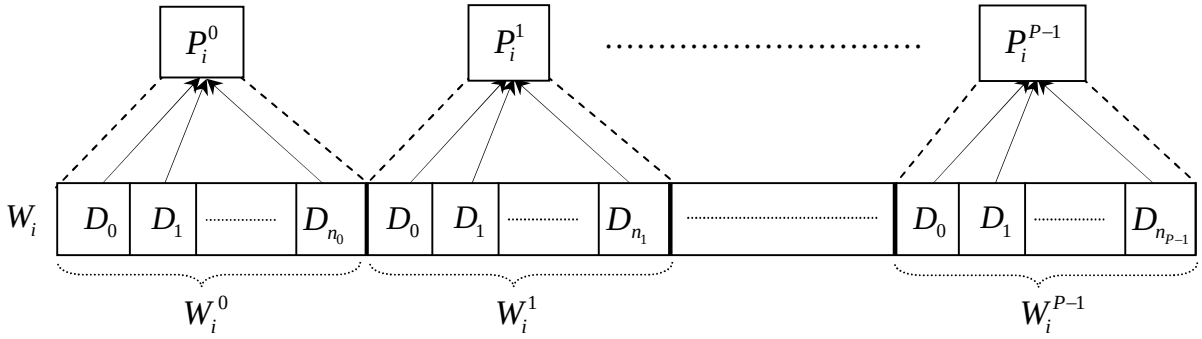


Figure 6. 4 Stratégie de distribution de données sur chaque site S_i

Sur chaque site S_i du réseau, le rafraîchissement de l'entrepôt W_i entraîne l'ajout d'un ensemble de nouvelles transactions. Cet ensemble de nouvelles transactions est distribué selon le mode Round Robin entre tous les processeurs de ce site, S_i ; si on note cet ensemble w_i alors les transactions de w_i sont réparties d'une façon cyclique entre tous les processeurs du site S_i. Ainsi, si les transactions T_0, T_1, …, T_{P-1} sont affectées aux processeurs P_0^i, P_1^i, …, P_{P-1}^i respectivement, alors les transactions T_P, T_{P+1}, …, T_{2P-1} seront affectées aux processeurs P_0^i, P_1^i, …, P_{P-1}^i respectivement, et ainsi de suite (voir la figure 6. 5). En conséquence, à chaque mise à jour d'un fragment W_i de l'entrepôt de données

W, l'ensemble w_i de nouvelles transactions ajoutées à cet fragment est réparti équitablement entre tous les processeurs du site S_i, en vue de son traitement en parallèle.

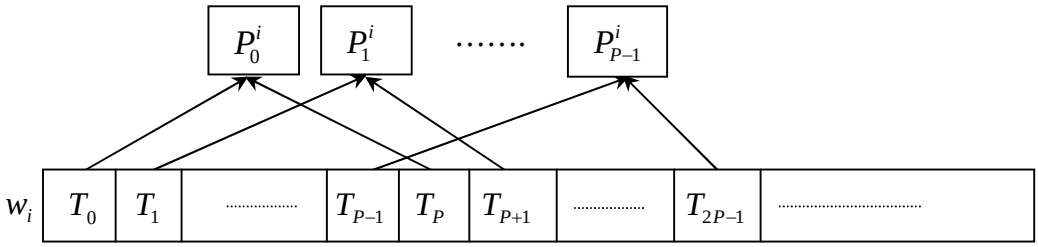


Figure 6. 5 Stratégie de distribution des transactions ajoutées au rafraîchissement d'un fragment W_i de l'entrepôt de données W

6. 5 Approche partitionnement et incrémentale d'extraction de règles d'association

L'extraction de règles d'association à partir des entrepôts de données est souvent basée sur ces deux approches : l'approche partitionnement (algorithme Partition [Agr 1993]) et l'approche incrémentale (algorithme Incremental Mining [Sav 1995]). En effet, l'approche partitionnement est la plus appropriée pour les supports de stockage d'énorme capacité, comme les entrepôts de données, puisque elle réduit le nombre de balayages de données nécessaires pour extraire toutes les règles d'associations pertinentes à deux balayages seulement. D'autre part, des mises à jour des données de l'entrepôt sont envisagés à chaque expiration d'un délai, appelé délai de rafraîchissement de l'entrepôt, invalide les connaissances extraites sur les anciennes données. Ainsi, l'utilisation d'une approche incrémentale est très nécessaire dans ce cas.

5. 5. 1 Approche partitionnement

L'algorithme Partition [Sav 1995] diffère de l'algorithme Apriori en termes de nombre de balayages de la base de données nécessaires pour extraire l'ensemble de tous les itemsets fréquents. En effet, par comparaison à l'algorithme Apriori qui nécessite un balayage de données à chaque itération, l'algorithme Partition génère l'ensemble des itemsets fréquents en deux balayages de données seulement. L'algorithme Partition s'exécute en deux phases : durant la première phase, la base de données est partitionnée horizontalement en n partitions disjointes, et chaque partition P_i est manipulée individuellement pour générer l'ensemble L^i des itemsets fréquents sur cette partition. En effet, l'algorithme Partition utilise le même principe que celui de l'algorithme Apriori pour générer les itemsets fréquents sur chaque partition. A la fin de la première phase, les ensembles des itemsets fréquents générés sur chaque partition sont fusionnés pour générer l'ensemble global C^G des itemsets candidats pour la deuxième phase ; ici, l'algorithme Partition utilise la propriété suivante : « un itemset

fréquents sur la totalité de la base de données doit être fréquents sur au moins une partition de celle-ci ». Durant la deuxième phase, le support de tous les itemsets candidats est recalculé sur la base de données complète, et l'ensemble L^G des itemsets fréquents est généré comme l'ensemble de tous les itemsets candidats qui ont satisfait le support minimum (*minsup*) spécifié par les utilisateurs. En conséquence, la base de données est parcourue complètement deux fois ; une fois à chaque phase de l'algorithme.

L'algorithme Partition peut être exécuté en parallèle en exploitant la capacité d'un certain nombre de processeurs. Les données sont distribuées à travers tous les processeurs et chaque processeur génère les itemsets fréquents pour son ensemble de données en parallèle. En conséquence, la génération des itemsets globalement fréquents nécessite l'échange des résultats intermédiaires entre les processeurs. L'algorithme parallèle Partition, bien qu'il été développé pour les multiprocesseurs, peut être adapté pour des environnements distribués où les données sont distribué à travers plusieurs base de données.

5. 5. 2 Approche incrémentale

Les algorithmes d'extraction de règles d'association, tels que Apriori et Partition, permettent de générer toutes les relations d'indépendances entre les attributs ou les items d'une base de données sous forme d'un ensemble de règles, appelées règles d'association. Mais les règles extraites de cette façon reflètent l'état actuel de la base de données et ces algorithmes ne sont pas appropries pour les bases de données dynamiques (mises à jour), comme les entrepôts de données. En effet, la mise à jour de l'entrepôt de données par des nouvelles transactions peut invalider les règles existantes ou introduire des nouvelles règles. La solution la plus simple pour rafraîchir l'ensemble de règles d'association est de recalculer l'ensemble des itemsets fréquents à chaque mise à jour de l'entrepôt de données. Mais cette solution est inefficace puisque la tâche de calcul des itemsets fréquents est très coûteuse. La meilleure façon d'extraire des règles d'association à partir des entrepôts de données est d'utiliser des techniques incrémentales. Ces techniques permettent d'exploiter efficacement l'ensemble des itemsets fréquents calculé antérieurement sans besoin de le recalculer à chaque mise à jour de l'entrepôt de données.

Agrawal et al. [Agr 1993] ont proposés l'algorithme Incremental Mining qui permet de mettre à jour l'ensemble des itemsets fréquents à chaque rafraîchissement de l'entrepôt de données. En effet, à la mise à jour de l'entrepôt de données des anciens itemsets fréquents peuvent devenir non fréquents ou des anciens itemsets non fréquents peuvent devenir fréquents sur l'entrepôt de données rafraîchi. L'algorithme Incremental Mining utilise le concept de frontière négative [Toi 1996] pour trouver les nouveaux itemsets fréquents, en évitant en maximum la tâche de recalculer des itemsets fréquents lorsque des nouvelles transactions sont ajoutées à l'entrepôt de données. La frontière négative consiste en tous les

itemsets qui sont candidats mais pas fréquents à chaque itération de l'algorithme Apriori. En effet, durant chaque itération de l'algorithme Apriori, l'ensemble des itemsets candidats C_k est calculé à partir de l'ensemble des itemsets fréquents L_{k-1}, et la frontière négative $FN(L_k)$ est l'ensemble de tous ces itemsets qui sont candidats à la $k^{\text{ième}}$ itération mais n'ont pas satisfait le support minimum (*minsup*) spécifié par les utilisateurs ; ainsi $FN(L_k) = C_k - L_k$. En conséquence, dans l'algorithme Incremental Mining, un balayage complet de l'entrepôt de données est exigé à chaque expansion de la frontière négative ; c'est-à-dire, si un itemsets en dehors de la frontière négative est ajouté à l'ensemble des itemsets fréquents ou à sa frontière négative [Val 2003].

6. 6 Algorithme proposé pour la génération des itemsets fréquents

Dans notre approche décrite dans la section 6. 8, la génération des itemsets localement fréquents sur chaque entrepôt W_i de chaque site S_i se base sur l'algorithme « Calcul_Itemsets_Fréquents », proposé dans cette section (algorithme 6. 1). Cet algorithme est très similaire à l'algorithme Partition, décrit dans la section 6. 5, pour lequel on apporte une amélioration. En effet, plusieurs améliorations ont été proposées pour l'algorithme Partition, comme par exemple réduire le nombre d'itemsets candidats en considérant les itemsets fréquents sur toutes les partitions sont fréquents sur la base de données. Dans l'algorithme « Calcul_Itemsets_Fréquents », on propose d'exécuter la première phase de l'algorithme Partition d'une façon incrémentale. Ainsi, cette démarche incrémentale nous nous permettrons de calculer, durant la première phase de l'algorithme, un sous ensemble d'itemsets fréquents et un sous ensemble d'itemsets de sa frontière négative, toute en réduisant le nombre d'itemsets candidats pour la deuxième phase de l'algorithme. En effet, l'ensemble des itemsets candidats est généré selon le même principe que l'algorithme Partition classique : « un itemset fréquent sur la base de données complète doit être fréquent sur au mois une partition de celle-ci ». Ainsi, durant la deuxième phase de l'algorithme, le support est recalculé pour ces itemsets candidats sur la base de données complète afin de générer le reste d'itemsets fréquents et le reste d'itemsets de sa frontière négative. Donc, comme l'algorithme Partition, l'algorithme proposé, « Calcul_Itemsets_Fréquents », s'exécute en deux phases suivantes :

La première phase

Durant la première phase de l'algorithme, la base de données est partitionnée horizontalement en n partitions disjointes, et chaque partition de données D est traitées individuellement pour calculer les ensembles d'itemsets suivants : l'ensemble L^D des itemsets fréquents sur la partition D et sa frontière négative $FN(L^D)$, l'ensemble L des

itemsets fréquents sur l'ensemble de transactions BD^+ et sa frontière négative $FN(L)$ et enfin, l'ensemble C des itemsets candidats sur l'ensemble de transactions BD^+. Noté que, l'ensemble BD^+ contient les transactions de la partition D et les transactions traitées avant le chargement de la partition D ; donc, $BD^+ = BD^- \cup D$, où l'ensemble BD^- contient les transactions traitées avant le chargement de la partition D.

La modification proposée par rapport à l'algorithme Partition est que, au traitement de chaque nouvelle partition de données D, l'algorithme proposé, en plus de traitement effectué pour calculer l'ensemble L^D des itemsets fréquents et sa frontière négative $FN(L^D)$, il met à jour l'ensemble L des itemsets fréquents, sa frontière négative $FN(L)$ et l'ensemble C des itemsets candidats, qui sont calculés sur l'ensemble de transactions traitées avant le chargement de la partition D (c'est-à-dire sur l'ensemble BD^-), de telle façon qu'ils soient ainsi sur l'ensemble de transactions $BD^+ = BD^- \cup D$. On dit alors que l'algorithme proposé met à jour ces connaissances (itemsets fréquents, itemsets non fréquents, itemsets candidats) sur l'ensemble de données traitées. La figure 6. 6 illustre se processus de traitement de la première phase de l'algorithme « Calcul_Itemsets_Fréquents ».

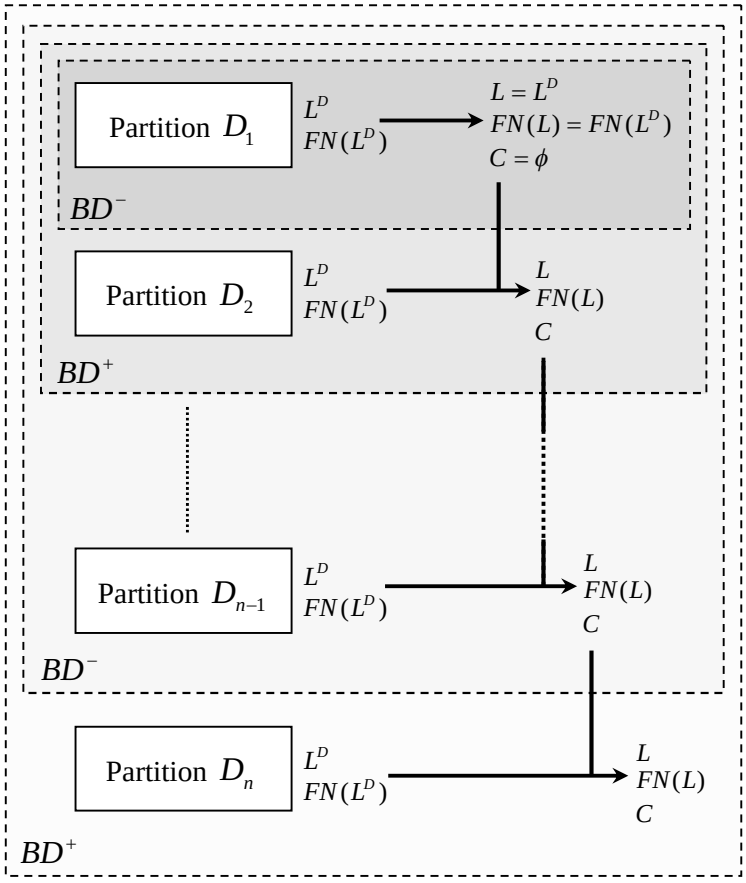


Figure 6. 6 Processus de traitement de la première phase

Afin d'expliquer le processus de mise à jour des ensembles L, $FN(L)$ et C, au traitement de chaque nouvelle partition de données D, nous déroulons ci-dessous un exemple explicatif.

La figure 6. 7 montre un exemple d'une table de transactions (BD) partitionnée en trois partitions D_1 , D_2 et D_3.

TID	*Items*	
T_1	A B C D	D_1
T_2	B C E	
T_3	A B C E	
T_4	B E	
T_5	A B C	D_2
T_6	B C D	
T_7	B D	
T_8	A B	
T_9	A B E F	D_3
T_{10}	A D E F	
T_{11}	A D F G	
T_{12}	A B E F	

Figure 6. 7 Exemple d'une table de transactions (BD)

On suppose que les partitions D_1 et D_2 sont traitées et la partition D_3 est chargée en mémoire centrale pour le traitement. La figure 6. 8 illustre l'ensemble L des itemsets fréquents, sa frontière négative $FN(L)$, l'ensemble C des itemsets candidats et l'ensemble d'items I , qui sont calculés sur l'ensemble de transactions traitées avant le chargement de la partition D_3 ; c'est-à-dire sur l'ensemble de transactions $BD^- = D_1 \cup D_2$. Noté que le support minimum (*minsup*) est égal à 30%.

Itemsets	A	B	C	D	E	AB	AC	BC	BE	ABC
Count	4	8	5	3	3	4	3	5	3	3

L

Itemsets	AE	CE	BCE	ABCE
Count	1	2	2	1

$FN(L)$

Itemsets	AD	BD	CD
Count	0	2	1

C

$I = \{A, B, C, D, E\}$

Figure 6. 8 Les ensembles L, $FN(L)$, C et I calculés sur l'ensemble de transactions $BD^- = D_1 \cup D_2$

Au chargement de chaque nouvelle partition de données D, les traitements suivants sont effectués sur cette partition :

1. Calculer l'ensemble L^D des itemsets fréquents sur la partition D, sa frontière négative $FN(L^D)$ et l'ensemble H des nouveaux items trouvés dans la partition D. Ce traitement est décrit par les étapes (1) et (2) de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1). La figure 6. 9 illustre les ensembles L^{D_3}, $FN(L^{D_3})$ et H calculés sur la partition D_3 de l'exemple de la figure 6. 7.

Itemsets	A	B	D	E	F	AB	AD	AE	AF	BE	BF	DF	EF	ABE	ABF	ADF	AEF	BEF	ABEF
Count	4	2	2	3	4	2	2	3	4	2	2	2	3	2	2	2	3	2	2

L^{D_3}

$FN(L^{D_3})$

Itemsets	G	BD	DE	ABDF	ADEF
Count	1	0	1	0	1

$H = \{F, G\}$

Figure 6. 9 Les ensembles L^{D_3} et $FN(L^{D_3})$ calculés sur la partition D_3

2. Mettre à jour les compteurs des itemsets de l'ensemble L et sa frontière négative $FN(L)$, calculés sur l'ensemble de transactions BD^-, sur la partition D et ainsi leurs nombres d'occurrences sur l'ensemble de transactions $BD^+ = BD^- \cup D$ est obtenu. Ce traitement est décrit par l'étape (3) de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1). La figure 6. 9 illustre les ensembles L et $FN(L)$ de la figure 6. 8, calculés sur l'ensemble de transactions $BD^- = D_1 \cup D_2$, dont les compteurs de leurs itemsets sont mises à jour sur la partition D_3.

Itemsets	A	B	C	D	E	AB	AC	BC	BE	ABC
Count	8	10	5	5	6	6	3	5	5	3

L

Itemsets	AE	CE	BCE	ABCE
Count	4	2	2	1

$FN(L)$

$I = \{A, B, C, D, E, F, G\}$

$H = \{F, G\}$

Figure 6. 10 Mise à jour des compteurs des itemsets des ensembles L et $FN(L)$ sur la partition D_3

Mettre à jour l'ensemble L des itemsets fréquents, sa frontière négative $FN(L)$ et l'ensemble C des itemsets candidats, qui sont calculés sur l'ensemble de transactions BD^-, de telle façon qu'ils soient ainsi sur l'ensemble de transactions $BD^+ = BD^- \cup D$. Ce traitement est décrit par les étapes (4), (5), (6) et (7) de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1). La figure 6. 11 illustre les

ensembles L, $FN(L)$ et C, obtenus après le traitement de la partition D_3 de l'exemple de la figure 6. 7.

Itemsets	A	B	C	D	E	F	AB	AE	AF	BC	BE
Count	8	10	5	5	6	4	6	4	4	5	5

L

Itemsets	AD	BD	CD	DE	ABE
Count	2	2	1	1	2

C

Itemsets	G	AC	BF	CE	DF	EF	ABC	ABF	ADF	AEF	BCE	BEF	ABCE	ABEF	ABDF	ADEF
Count	1	3	2	2	2	2	3	2	2	3	2	2	1	2	0	1

$FN(L)$

Figure 6. 11 Les ensembles L, $FN(L)$ et C calculés sur l'ensemble de transactions $BD^+ = D_1 \cup D_2 \cup D_3$

Au chargement de chaque nouvelle partition de données D, la mise à jour des ensembles L, $FN(L)$ et C est effectuée selon ces deux situations suivantes :

La 1^{ière} Situation

En résulte de la mise à jour des compteurs des itemsets de l'ensemble L et sa frontière négative $FN(L)$ sur la partition D deux situations : des itemsets de l'ensemble L, qui sont fréquents sur l'ensemble de transactions BD^-, peuvent devenir non fréquents sur l'ensemble de transactions $BD^+ = BD^- \cup D$, et des itemsets de l'ensemble $FN(L)$, qui sont non fréquents sur l'ensemble de transactions BD^-, peuvent devenir fréquents sur l'ensemble de transactions $BD^+ = BD^- \cup D$. En conséquence, les traitements suivants sont effectués sur ces deux ensembles, L et $FN(L)$:

1. Les itemsets de l'ensemble L qui sont devenus non fréquents sur l'ensemble de transactions $BD^+ = BD^- \cup D$ seront supprimés de l'ensemble L et ajoutés à l'ensemble $FN(L)$. Comme elle montre la figure 6. 10, la mise à jour des compteurs des itemsets de l'ensemble L, présenté sur la figure 6. 8, sur la partition D_3 a rendue les itemsets AC et ABC non fréquents sur l'ensemble de transactions $BD^+ = D_1 \cup D_2 \cup D_3$; Alors, comme elle illustre la figure 6. 12, ces deux itemsets déplacent de l'ensemble L à l'ensemble $FN(L)$. Ce traitement est décrit par l'étape (4) de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1) ; il peut être résumé comme suit :

$$\forall\, x \in L \begin{cases} \textbf{Si} \;\; x.count \geq \text{minsup} \times M \;\; \textbf{Alors} \;\; x \in L \\ \textbf{Si} \;\; x.count < \text{minsup} \times M \;\; \textbf{Alors} \;\; x \in FN(L) \end{cases}$$

Où M est le nombre de transaction de l'ensemble $BD^+ = BD^- \cup D$.

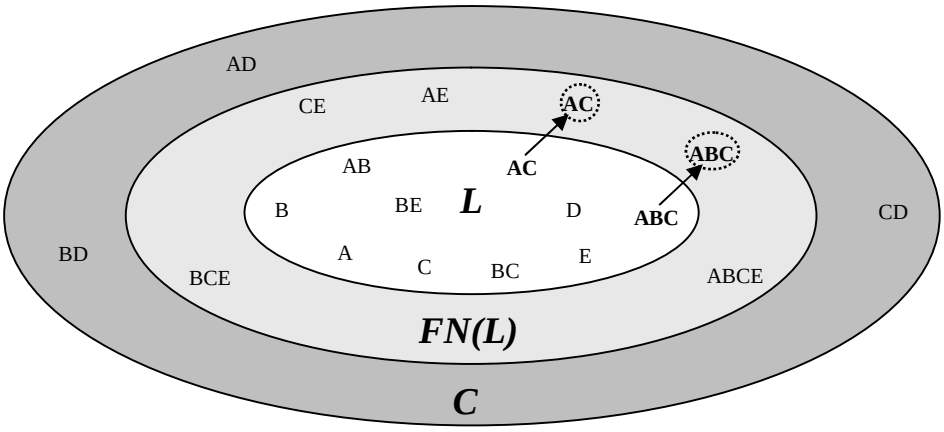


Figure 6. 12 Le 1ière traitement pour la mise à jour des ensembles L, $FN(L)$ et C

2. Les itemsets de l'ensemble $FN(L)$ qui sont devenus fréquents sur l'ensemble de transactions $BD^+ = BD^- \cup D$ seront supprimés de l'ensemble $FN(L)$ et ajoutés à l'ensemble L. Comme elle montre la figure 6. 10, la mise à jour des compteurs des itemsets de l'ensemble $FN(L)$, présenté sur la figure 6. 8, sur la partition D_3 a rendue l'itemset AE fréquents sur l'ensemble de transactions $BD^+ = D_1 \cup D_2 \cup D_3$; Alors, comme elle illustre la figure 6. 13, cet itemset déplace de l'ensemble $FN(L)$ à l'ensemble L. Ce traitement est décrit par l'étape (5) de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1) ; il peut être résumé comme suit :

$$\forall\, x \in FN(L) \begin{cases} \textbf{Si}\ \ x.count \geq \text{minsup} \times M \quad \textbf{Alors} \quad x \in L \\ \textbf{Si}\ \ x.count < \text{minsup} \times M \quad \textbf{Alors} \quad x \in FN(L) \end{cases}$$

Où M est le nombre de transaction de l'ensemble $BD^+ = BD^- \cup D$.

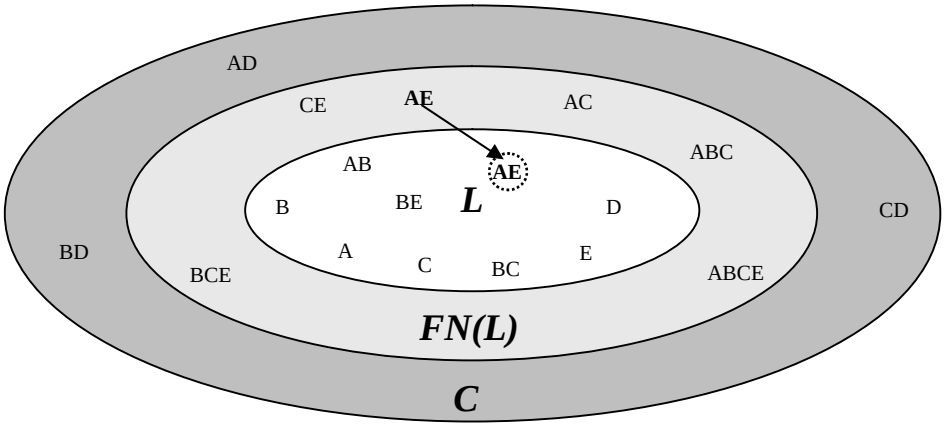


Figure 6. 13 Le 2ième traitement pour la mise à jour des ensembles L, $FN(L)$ et C

La 2^{ième} Situation

Lors de traitement de chaque nouvelle partition D, l'ensemble H des nouveaux items trouvés sur cette partition est généré. Comme elle décrit l'étape (1) de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1), si C_1 est l'ensemble d'items trouvés sur la partition D et I l'ensemble d'items trouvés sur les partitions traitées avant le chargement de la partition D alors $H = C_1 - I$ constitue l'ensemble des nouveaux items trouvés sur la partition D. En effet, les items de cet ensemble sont utilisés dans le processus de distribution des itemsets de l'ensembles L^D et sa frontière négative $FN(L^D)$ entre les ensembles L, $FN(L)$ et C.

Nous avons remarqués que les itemsets de l'ensemble L^D, ou de sa frontière négative $FN(L^D)$, qui contiennent au moins un items de l'ensemble H n'appartiennent pas aux transactions traitées avant le chargement de la partition D. En conséquence, leurs nombres d'occurrences calculés sur la partition D est égale à leurs nombres d'occurrences sur l'ensemble de transactions $BD^+ = BD^- \cup D$; ainsi leurs supports sur l'ensemble de transactions BD^+ est connus. En effet, selon que ces itemsets soient fréquents ou non fréquents sur l'ensemble de transactions BD^+, ils seront ajoutés à l'ensembles L ou $FN(L)$ respectivement. Cependant, les itemsets de l'ensemble L^D, ou de sa frontière négative $FN(L^D)$, qui ne contiennent aucun items de l'ensemble H, pour lesquels leur support est inconnu sur l'ensemble de transactions BD^+, seront ajoutés à l'ensemble des itemsets Candidats, C.

L'idée de base de l'algorithme proposé (« Calcul_Itemsets_Fréquents ») est de générer, au traitement de chaque nouvelle partition de données D, les nouveaux itemsets trouvés sur la partition D et de suivre leurs apparition sur le reste de transactions de la base de données ; un itemset est dit nouveau s'il contient au moins un items de l'ensemble H. Ainsi, Ces nouveaux itemsets sont répartis sur les ensembles L et $FN(L)$, selon qu'ils soient fréquents ou non fréquents respectivement.

Le processus de distribution des itemsets de l'ensemble L^D entre les ensembles L, $FN(L)$ et C peut être expliqué comme suit :

1. Les itemsets de l'ensemble L^D qui contiennent au moins un items de l'ensemble H, si ils sont fréquents sur l'ensemble de transactions $BD^+ = BD^- \cup D$ alors ils seront ajoutés à l'ensemble L. Comme elle montre la figure 6. 14, les itemsets F et AF de l'ensemble L^{D_3}, présenté sur la figure 6. 9, sont ajoutés à l'ensemble L puisque ils contiennent un items de l'ensemble H, $F \in H$, et ils sont fréquents sur l'ensemble de

transactions $BD^+ = D_1 \cup D_2 \cup D_3$. Ce traitement est décrit sur l'étape (6) de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1), il peut être résumé comme suit :

$$\forall\ x \in L^D \quad \textbf{Si}\ x \cap H \neq \phi\ \wedge\ x.count \geq \text{minsup} \times M\ \textbf{alors}\ x \in L$$

Où M est le nombre de transactions de l'ensemble $BD^+ = BD^- \cup D$.

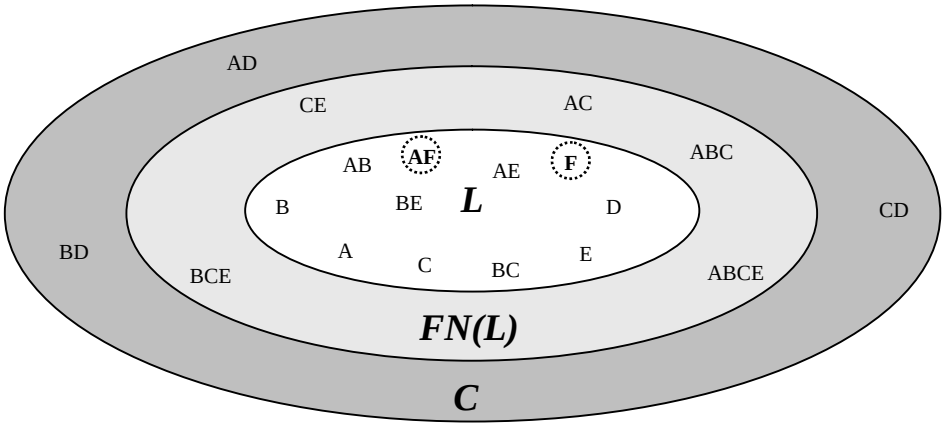


Figure 6. 14 Le 3ième traitement pour la mise à jour des ensembles L, $FN(L)$ et C

2. Les itemsets de l'ensemble L^D qui contiennent au moins un items de l'ensemble H, si ils sont non fréquents sur l'ensemble de transactions $BD^+ = BD^- \cup D$ alors ils seront ajoutés à l'ensemble $FN(L)$. Comme elle montre la figure 6. 15, les itemsets BF, DF, EF, ABF, ADF, AEF, BEF et ABEF de l'ensemble L^{D_3}, présenté sur la figure 6. 9, sont ajoutés à l'ensemble $FN(L)$ puisqu'ils contiennent un items de l'ensemble H, $\text{F} \in H$, et ils sont non fréquents sur l'ensemble de transactions $BD^+ = D_1 \cup D_2 \cup D_3$. Ce traitement est décrit sur l'étape (6) de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1), il peut être résumé comme suit :

$$\forall\ x \in L^D \quad \textbf{Si}\ x \cap H \neq \phi\ \wedge\ x.count < \text{minsup} \times M\ \textbf{alors}\ x \in FN(L)$$

Où M est le nombre de transactions de l'ensemble $BD^+ = BD^- \cup D$.

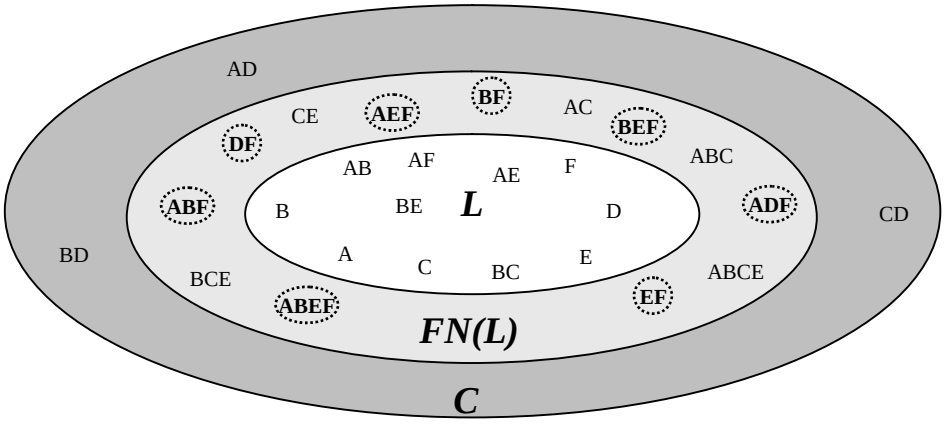


Figure 6. 15 Le 4ième traitement pour la mise à jour des ensembles L, $FN(L)$ et C

3. Le reste d'itemsets de l'ensemble L^D qui ne contiennent aucun items de l'ensemble H, il seront ajoutés à l'ensemble C des itemsets candidats à condition qu'ils n'appartiennent pas ni à l'ensemble L ni à sa frontière négative $FN(L)$. Comme elle montre la figure 6. 16, les itemsets AD et ABE de l'ensemble L^{D_3}, présenté sur la figure 6. 9, sont ajoutés à l'ensemble C puisque ils ne contiennent pas des items de l'ensemble H et ils n'appartiennent ni à l'ensemble L ni à l'ensemble $FN(L)$. Ce traitement est décrit sur l'étape (6) de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1), il peut être résumé comme suit :

$$\forall\, x \in L^D \quad \textbf{Si}\ x \cap H = \phi \ \wedge\ x \notin L \ \wedge\ x \notin FN(L)\ \textbf{alors}\ x \in C$$

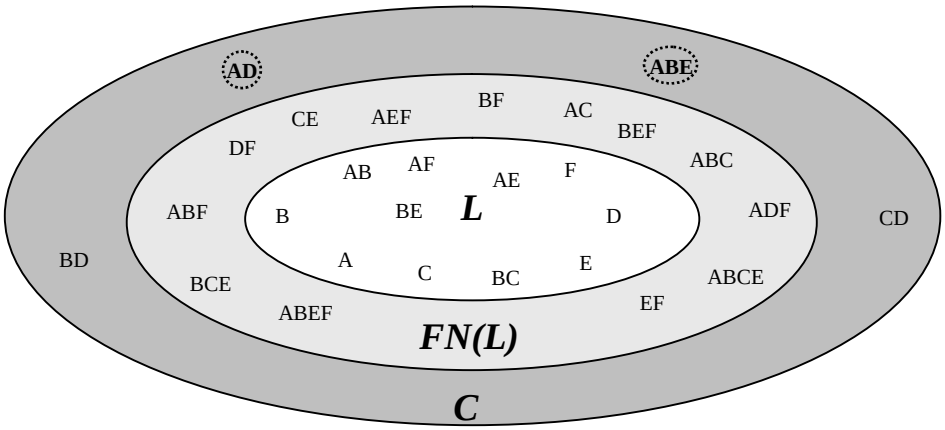


Figure 6. 16 Le 5ième traitement pour la mise à jour des ensembles L, $FN(L)$ et C

Le processus de distribution des itemsets de l'ensemble $FN(L^D)$ entre les ensembles L, $FN(L)$ et C peut être expliqué comme suit :

1. Les itemsets de l'ensemble $FN(L^D)$ qui contiennent au moins un items de l'ensemble H ils seront ajoutés à l'ensemble $FN(L)$, puisqu'ils sont non fréquents sur la partition D alors ils seront non fréquents sur l'ensemble de transactions $BD^+ = BD^- \cup D$. Comme elle montre la figure 6. 17, les itemsets G, ABDF et ADEF de l'ensemble $FN(L^{D_3})$, présenté sur la figure 6. 9, sont ajoutés à l'ensemble $FN(L)$ puisque ils contiennent un items de l'ensemble H, $F \in H$ et $G \in H$, et comme ils sont non fréquents sur la partition D_3 alors ils seront non fréquents sur l'ensemble de transactions $BD^+ = D_1 \cup D_2 \cup D_3$. Ce traitement est décrit sur l'étape (7) de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1), il peut être résumé comme suit :

$$\forall\, x \in FN(L^D) \quad \textbf{Si}\ x \cap H \neq \phi\ \textbf{alors}\ x \in FN(L)$$

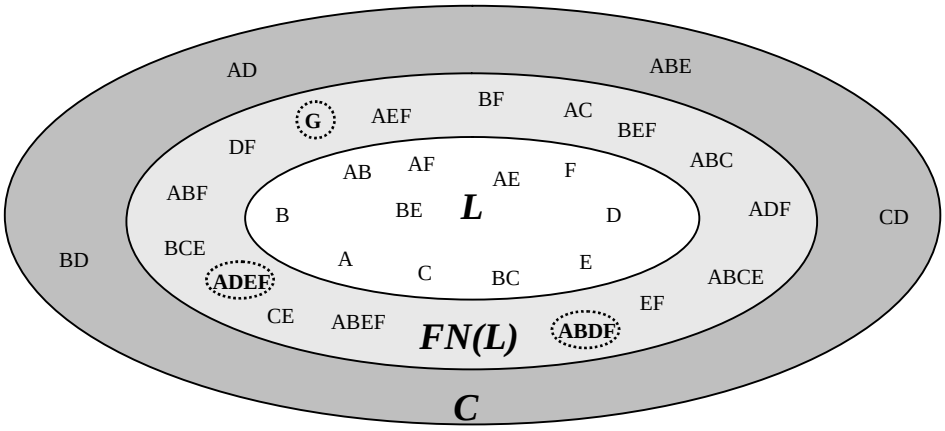


Figure 6. 17 Le 6ième traitement pour la mise à jour des ensembles L, $FN(L)$ et C

2. Le reste d'itemsets de l'ensemble $FN(L^D)$ qui ne contiennent aucun items de l'ensemble H, il seront ajoutés à l'ensemble C des itemsets candidats à condition qu'ils n'appartiennent pas ni à l'ensemble L ni à sa frontière négative $FN(L)$. Comme elle montre la figure 6. 18, les itemsets BD et DE de l'ensemble $FN(L^{D_3})$, présenté sur la figure 6. 9, sont ajoutés à l'ensemble C puisque ils ne contiennent pas des items de l'ensemble H et ils n'appartiennent ni à l'ensemble L ni à l'ensemble $FN(L)$. Ce traitement est décrit sur l'étape (7) de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1), il peut être résumé comme suit :

$$\forall\, x \in FN(L^D) \quad \textbf{Si}\ x \cap H = \phi \ \wedge\ x \notin L \ \wedge\ x \notin FN(L)\ \textbf{alors}\ x \in C$$

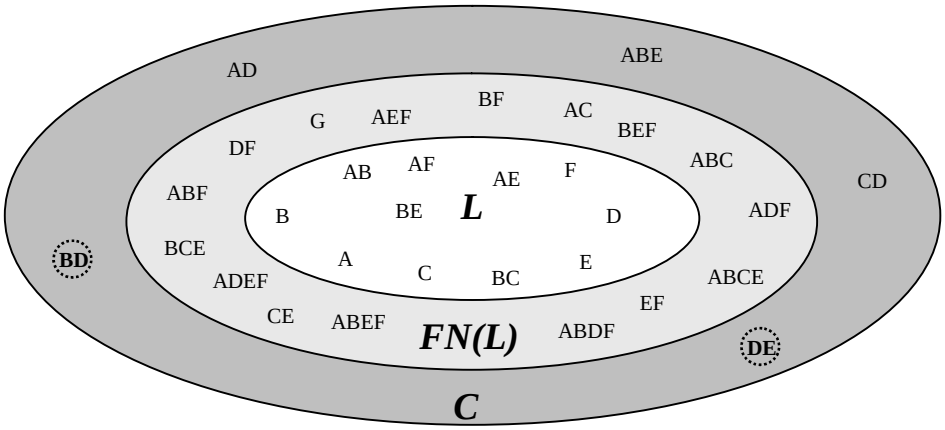


Figure 6. 18 Le 7ième traitement pour la mise à jour des ensembles L, $FN(L)$ et C

La deuxième phase

Le traitement incrémental effectué durant la première phase de l'algorithme « Calcul_Itemsets_Fréquents » (algorithme 6. 1) permet de calculer un sous ensemble d'itemsets fréquents et un sous ensemble de sa frontière négative, et ainsi, la deuxième phase à eu lieu pour rattraper le reste d'itemsets fréquents et le reste d'itemsets non fréquents de sa frontière négative qui ne sont pas générés lors de la première phase de l'algorithme. En effet,

durant la deuxième phase, la base de données est parcourue complètement pour recalculer le support des itemsets candidats de l'ensemble C, et selon que son supports soit supérieur ou inférieur à un certain seuil (*minsup*) spécifié par les utilisateurs, les itemsets de l'ensemble C seront ajoutés à l'ensemble L ou à sa frontière négative $FN(L)$ respectivement.

Algorithme 6. 1 Calcul_Itemsets_Fréquents

Début

// Initialisation

$L = \phi$; // L'ensemble des itemsets fréquents

$FN(L) = \phi$; // L'ensemble des itemsets de la frontière négative

$C = \phi$; // L'ensemble des itemsets candidats

$M = 0$; // Le nombre de transactions traitées

$I = \phi$; // L'ensemble d'items traités

// Phase I

n = Partitionner (BD) ; // Partitionner la base de données BD en n partitions disjointes

Pour $i = 1$ à n **Faire**

Lire_Partition (D , BD) ;

$M = M + |D|$; // $|D|$ est égale au nombre de transactions de la partition D

(1) $C_1 = \phi$;

Pour toutes transactions $t \in D$ **Faire**

$C_1 = C_1 \cup \{ x / x \subset t \wedge |x| = 1 \}$;

Pour tous items $x \in C_1 \wedge x \subset t$ **Faire**

x .Count ++;

Fin Pour ;

Fin Pour ;

$H = C_1 - I$; // H est l'ensemble des nouveaux items trouvés dans la partition D

$I = I \cup H$;

(2) $L^D = \{ x / x \in C_1 \wedge x.\text{Count} \geq (minsup \times |D|) \}$;

$FN(L^D) = \{ x / x \in C_1 \wedge x.\text{Count} < (minsup \times |D|) \}$;

Pour $k = 2$, $L_{k-1} \neq \phi$, k ++ **Faire**

// Générer les k-itemsets candidats

C_k = Apriori_gen (L_{k-1}) ;

Pour toutes transactions $t \in D$ **Faire**

Pour tous itemsets $x \in C_k \wedge x \subset t$ **Faire**

x .Count ++;

Fin Pour ;

Fin pour ;

$L^D = L^D \cup \{ x / x \in C_k \wedge x.\text{Count} \geq (minsup \times |D|) \}$;

$FN^D = FN^D \cup \{ x / x \in C_k \wedge x.\text{Count} < (minsup \times |D|) \}$;

Fin Pour ;

(3) **Pour** toutes transactions $t \in D$ **Faire**

Pour tous itemsets $x \in L \wedge x \subset t$ ou $x \in FN(L) \wedge x \subset t$ **Faire**

x.Count ++ ;

Fin Pour ;

Fin Pour ;

(4) **Pour** tous itemsets $x \in L$ **Faire**

Si x.Count < $minsup \times M$ **Alors**

$L = L - \{ x \}$;

$FN(L) = FN(L) \cup \{ x \}$;

Fin si ;

Fin Pour ;

(5) **Pour** tous itemsets $x \in FN(L)$ **Faire**

Si x.Count $\geq minsup \times M$ **Alors**

$FN(L) = FN(L) - \{ x \}$;

$L = L \cup \{ x \}$;

Fin si ;

Fin Pour ;

(6) **Pour** tous itemsets $x \in L^D$ **Faire**

Si $x \cap H \neq \phi$ **Alors**

Si x.Count $\geq minsup \times M$ **Alors**

$L = L \cup \{ x \}$;

Sinon

$FN(L) = FN(L) \cup \{ x \}$;

Fin si ;

Sinon

Si $x \notin L \wedge x \notin FN(L)$ **Alors**

$C = C \cup \{ x \}$;

Fin si ;

Fin si ;

Fin Pour ;

(7) **Pour** tous itemsets $x \in FN(L^D)$ **Faire**

Si $x \cap H \neq \phi$ **Alors**

$FN(L) = FN(L) \cup \{ x \}$;

Sinon

Si $x \notin L \wedge x \notin FN(L)$ Alors

$C = C \cup \{ x \}$;

Fin si ;

Fin si ;
Fin Pour ;
Fin Pour ;

// Phase II

Pour $i = 1$ à n **Faire**
Lire_Partition (D , BD) ;
Pour toutes transactions $t \in D$ **Faire**
Pour tous itemsets $x \in C \wedge x \subset t$ **Faire**
x .Count ++ ;
Fin Pour ;
Fin Pour ;
Fin Pour ;
$L = L \cup \{ x / x \in C \wedge x.\text{Count} \geq minsup \times M \}$;
$FN(L) = FN(L) \cup \{ x / x \in C \wedge x.\text{Count} < minsup \times M \}$;
Fin.

6. 7 Algorithme proposé pour la mise à jour des itemsets fréquents

L'algorithme proposé dans la section précédente permet de générer l'ensemble des itemsets fréquents et sa frontière négative dans une base de données ou un entrepôt de données. Dans cette section, nous proposons un algorithme incrémental, appelé « Mise_à_jour_Itemsets_Fréquents », qui sera exploité pour la mise à jour des itemsets fréquents à chaque rafraîchissement de l'entrepôt de données. En effet, dans notre approche décrite dans la section 6. 8, la mise à jour des itemsets fréquents sur chaque site S_i se base sur cet algorithme. Comme l'algorithme Incremental Mining, décrit dans la section 6. 5, l'algorithme « Mise_à_jour_Itemsets_Fréquents » exploite la frontière négative pour décider quand il faut un parcourt complet sur les données de l'entrepôt. Par comparaison à l'algorithme Incremental Mining qui nécessite un balayage de l'entrepôt à chaque fois que des itemsets en dehors de la frontière négative sont ajoutés à l'ensemble des itemsets fréquents ou à sa frontière négative, l'algorithme « Mise_à_jour_Itemsets_Fréquents » nécessite un balayage de l'entrepôt à chaque fois qu'il trouve sur la partition ajoutée des itemsets fréquents en dehors de la frontière négative qui ne contiennent pas des items de l'ensemble H , où H est l'ensemble des nouveaux items trouvés sur la partition ajoutée.

À chaque rafraîchissement de l'entrepôt de données, l'algorithme « Mise_à_jour_Itemsets_Fréquents » traite la partition ajoutée d'une façon identique au traitement effectué par l'algorithme « Calcul_Itemsets_Fréquents » sur une partition de l'entrepôt de données. Ainsi, à chaque rafraîchissement de l'entrepôt de données, le traitement effectué par l'algorithme « Mise_à_jour_Itemsets_Fréquents » sur la partition ajoutée peut être résumé comme suit :

1. Calculer l'ensemble H des nouveaux items trouvés sur la partition ajoutée w ; voir l'étape (1) de l'algorithme « Mise_à_jour_Itemsets_Fréquents » (algorithme 6. 2).
2. Calculer l'ensemble L^w des itemsets fréquents et sa frontière négative $FN(L^w)$ sur la partition ajoutée w ; voir l'étape (2) de l'algorithme « Mise_à_jour_Itemsets_Fréquents » (algorithme 6. 2).
3. Mettre à jour les compteurs des itemsets de l'ensemble L et les compteurs des itemsets de l'ensemble $FN(L)$ sur la partition ajoutée w ; voir l'étape (3) de l'algorithme « Mise_à_jour_Itemsets_Fréquents » (algorithme 6. 2).
4. Mettre à jour l'ensemble L des itemsets fréquents et sa frontière négative $FN(L)$ sur l'entrepôt rafraîchi $W^+ = W \cup w$; voir les étapes (4), (5), (6) et (7) de l'algorithme « Mise_à_jour_Itemsets_Fréquents » (algorithme 6. 2).

Dans l'algorithme « Mise_à_jour_Itemsets_Fréquents », l'ensemble des itemsets fréquents et sa frontière négative peut être augmenté au rafraîchissement de l'entrepôt de données par l'ajout des itemsets en dehors de la frontière négative. En effet, les itemsets de l'ensemble L^w et sa frontière négative $FN(L^w)$ qui contiennent au moins un item de l'ensemble H sont ajoutés à l'ensemble L ou à sa frontière négative $FN(L)$ selon qu'ils soient fréquent ou non fréquents respectivement sur l'entrepôt de données rafraîchi $W^+ = W \cup w$. Cependant, les itemsets de l'ensemble L^w et sa frontière négative $FN(L^w)$ qui ne contiennent pas des items de l'ensemble H sont accumulés dans l'ensemble des itemsets candidats C s'ils sont en dehors de la frontière négative. Ainsi, l'entrepôt de données est parcouru complètement pour calculer le nombre d'occurrences des itemsets de l'ensemble C à chaque fois que des itemsets de l'ensemble L^w sont ajoutés à l'ensemble C ; c'est-à-dire s'il existe dans L^w des itemsets en dehors de la frontière négative qui ne contiennent pas des items de l'ensemble H. Donc, comme il décrit l'étape (8) de l'algorithme « Mise_à_jour_Itemsets_Fréquents », l'entrepôt de données est parcouru complètement pour calculer le nombre d'occurrences des itemsets de l'ensemble C à chaque fois la condition $L^w - (L \cup FN(L)) \neq \phi$ est vérifiée.

Algorithme 6. 2 Mise_à_jour_Itemsets_Fréquents

Début

// L dénote l'ensemble des itemsets fréquents sur l'entrepôt de données W

// $FN(L)$ dénote l'ensemble des itemsets de la frontière négative dans l'entrepôt de données W

// $C = \phi$ dénote l'ensemble des itemsets candidats calculé sur les partition ajoutées

Tant que (Vrai) **Faire**

Lire (w) ; // w dénote la partition de données ajoutée

$M = M + |w|$; // M dénote la taille de l'entrepôt de données

(1) Calculer H ; // H est l'ensemble des nouveaux items trouvés dans la partition ajoutée w

$I = I \cup H$

(2) Calculer L^w, $FN(L^w)$;

(3) **Pour** toutes transactions $t \in w$ **Faire**

Pour tous itemsets $x \in L \wedge x \subset t$ ou $x \in FN(L) \wedge x \subset t$ **Faire**

x .Count ++ ;

Fin Pour ;

Fin Pour ;

(4) **Pour** tous itemsets $x \in L$ **Faire**

Si x .Count < $minsup \times M$ **Alors**

$L = L - \{ x \}$;

$FN(L) = FN(L) \cup \{ x \}$;

Fin si ;

Fin Pour ;

(5) **Pour** tous itemsets $x \in FN(L)$ **Faire**

Si x .Count $\geq$ $minsup \times M$ **Alors**

$FN(L) = FN(L) - \{ x \}$;

$L = L \cup \{ x \}$;

Fin si ;

Fin Pour ;

(6) **Pour** tous itemsets $x \in L^w$ **Faire**

Si $x \cap H \neq \phi$ **Alors**

Si x .Count $\geq$ $minsup \times M$ **Alors**

$L = L \cup \{ x \}$;

Sinon

$FN(L) = FN(L) \cup \{ x \}$;

Fin si ;

Sinon

Si $x \notin L \wedge x \notin FN(L)$ **Alors**

$C = C \cup \{ x \}$;

Fin si ;

Fin si ;

Fin Pour ;

(7) **Pour** tous itemsets $x \in FN(L^w)$ **Faire**

Si $x \cap H \neq \phi$ **Alors**

$FN(L) = FN(L) \cup \{ x \}$;

Sinon

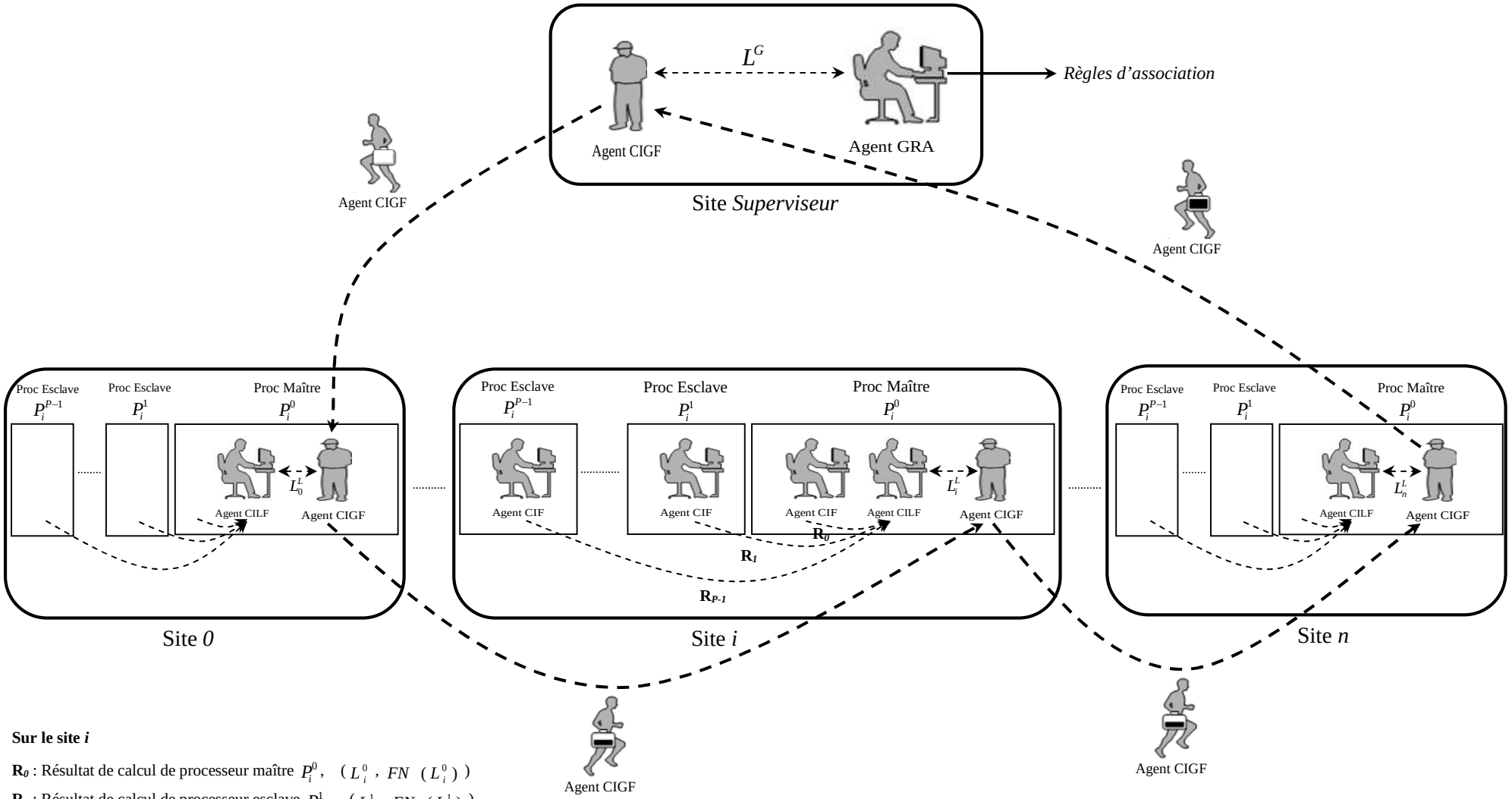
Si $x \notin L \wedge x \notin FN(L)$ Alors

$C = C \cup \{ x \}$;
Fin si ;
Fin si ;
Fin Pour ;
(8) **Si** $L^w - (L \cup FN(L)) \neq \phi$ **Alors**
Pour $i = 1$ à n **Faire**
Lire_Partition (D , W^+) ; // $W^+ = W \cup w$ est l'entrepôt de données rafraîchi
Pour toutes transactions $t \in D$ **Faire**
Pour tous itemsets $x \in C \wedge x \subset t$ **Faire**
x .Count ++ ;
Fin Pour ;
Fin Pour ;
Fin Pour ;
$L = L \cup \{ x / x \in C \wedge x.\text{Count} \geq minsup \times M \}$;
$FN(L) = FN(L) \cup \{ x / x \in C \wedge x.\text{Count} < minsup \times M \}$;
$C = \phi$;
Fin Si ;
Fin Tant que ;
Fin.

6. 8 Approche parallèle et distribuée d'extraction de règles d'association

Dans cette section, nous proposons une approche parallèle et distribuée d'extraction de règles d'association basée sur des agents mobiles et stationnaires, qui coopèrent pour calculer les itemsets localement fréquents sur chaque site du réseau, les itemsets globalement fréquents sur la totalité des données de l'entrepôt et enfin, l'ensemble de règles d'association. En outre, l'approche proposée dans cette section prend en considération le cas de mise à jour de l'entrepôt de données en proposant une démarche permettant de mettre à jour les itemsets localement fréquents, les itemsets globalement fréquents et l'ensemble de règles d'association à chaque rafraîchissement de l'entrepôt de données. Noté que, cette approche assume l'architecture et la configuration du réseau décrite dans la section 6. 3, et la stratégie de distribution de données décrite dans la section 6. 4.

Dés sont apparition, les agents mobiles constituent une solution efficace pour la résolution des problèmes distribués, qui nécessite la coopération entre plusieurs entités logiciels. En effet, les agents mobiles permettre de réduire la latence du réseau ; au lieu de déplacer les données pour leurs traitement centralisé, le concept d'agents mobiles offre la possibilité de traiter les données la où se trouve en déplaçant le code de l'agent et son état d'exécution. Dans l'annexe, une présentation de quelques terminologies liées aux agents mobiles et aux aglets (la plateforme la plus populaire des agents mobiles) est réalisée.



Sur le site *i*

$\mathbf{R_0}$: Résultat de calcul de processeur maître P_i^0, $(L_i^0, FN(L_i^0))$

$\mathbf{R_1}$: Résultat de calcul de processeur esclave P_i^1, $(L_i^1, FN(L_i^1))$

$\mathbf{R_{P-1}}$: Résultat de calcul de processeur esclave P_i^{P-1}, $(L_i^{P-1}, FN(L_i^{P-1}))$

Figure 6. 19 Démarche d'extraction de règles d'association dans l'approche proposée

6. 8. 1 Génération des itemsets localement fréquents

Sur chaque site S_i du réseau, l'extraction de tous les itemsets localement fréquents sur le fragment W_i de l'entrepôt de données W s'effectue d'une façon parallèle en exploitant la puissance de calcul et la capacité mémoire de P processeurs. En effet, à chaque processeur P_i^j (maître et esclaves) du site S_i est assigné un sous espace de données du W_i, noté W_i^j, et génère un ensemble d'itemsets localement fréquents sur cet sous espace de données, noté L_i^j, et sa frontière négative , noté $FN(L_i^j)$. La figure ci-dessous illustre cette démarche d'extraction d'itemsets localement fréquents sur chaque site S_i du réseau.

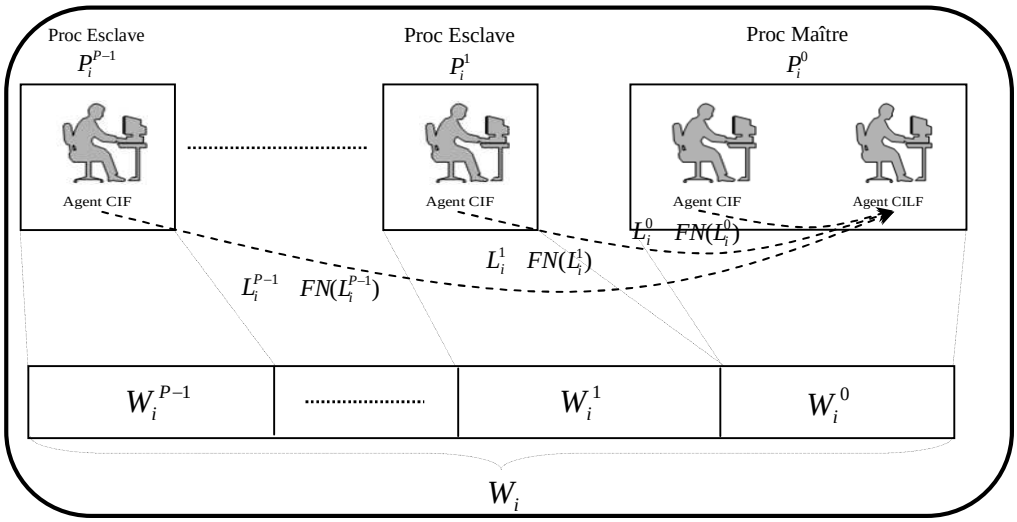


Figure 6. 20 Démarche d'extraction d'itemsets localement fréquents sur un site S_i du réseau

Sur chaque processeur P_i^j (maître et esclaves) du chaque site S_i, la tâche de génération des itemsets localement fréquents L_i^j et sa frontière négative $FN(L_i^j)$ est réalisée par un agent stationnaire, appelé « Agent Calcul_Itemsets_Fréquents » (ou encore « Agent CIF »). Le principe de fonctionnement de cet agent peut être résumé en trois étapes suivantes :

- *Initialisation* : Dans cette étape, l'ensemble des itemsets fréquents L_i^j et sa frontière négative $FN(L_i^j)$ sont initialisés ($L_i^j = \phi$ et $FN(L_i^j) = \phi$).
- *Traitement* : Cette étape constitue l'étape principale du processus de traitement de l'« Agent Calcul_Itemsets_Fréquents », puisque la tâche de génération des itemsets fréquents L_i^j et sa frontière négative $FN(L_i^j)$ est effectuée durant cette étape. En effet, pour calculer les ensembles L_i^j et $FN(L_i^j)$ sur l'espace de données W_i^j, l'« Agent Calcul_Itemsets_Fréquents » fait appel à la fonction « Calcul_Itemsets_Fréquents (L_i^j,

$FN(L_i^j)$) » (ou « fonction CIF ») qui porte l'algorithme « Calcul_Itemsets_Fréquents » décrit dans la section 6. 6 (voir la figure 6. 21).

- *Transmission* : Une fois l'ensemble des itemsets fréquents L_i^j et sa frontière négative $FN(L_i^j)$ sont calculées, l'« Agent Calcul_Itemsets_Fréquents » transmet d'une façon asynchrone ces résultats de calcul (c'est-à-dire L_i^j et $FN(L_i^j)$) à l'agent stationnaire « Agent Calcul_Itemsets_Localement_Fréquents » (ou encore « Agent CILF »), qui s'occupe de les combinés afin de générer l'ensemble L_i^L des itemsets localement fréquents et sa frontière négative $FN(L_i^L)$ sur le site S_i (ou sur le fragment W_i de l'entrepôt de données W).

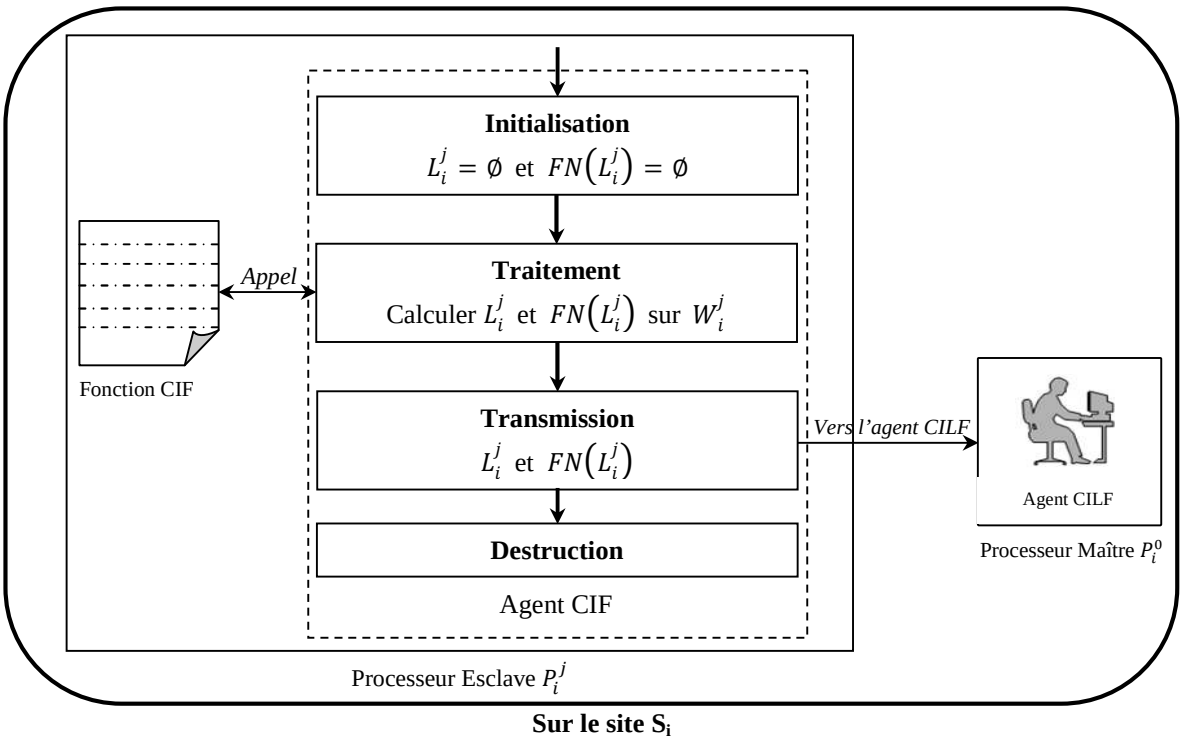


Figure 6. 21 Principe de fonctionnement de l'« Agent Calcul_Itemsets_Fréquents »

L'algorithme ci-dessous décrit le code de l'« Agent Calcul_Itemsets_Fréquents ».

Algorithme 6. 3 Code « Agent Calcul_Itemsets_Fréquents »
Début

$L_i^j = \phi$;

$FN(L_i^j) = \phi$;

Calcul_Itemsets_Fréquents (L_i^j , $FN(L_i^j)$) ;

Transmission de L_i^j et $FN(L_i^j)$ à l'« Agent Calcul_Itemsets_Localement_Fréquents » ;

Fin.

Sur chaque processeur maître P_i^0 de chaque site S_i, l'« Agent Calcul_Itemsets_Localement_Fréquents » attend la réception de tous les ensembles L_i^j et $FN(L_i^j)$, calculés par les « Agents Calcul_Itemsets_Fréquents » de tous les processeurs P_i^j (maître et esclaves) de même site S_i, et les combines ensuite en ensembles L_i^L et $FN(L_i^L)$ des itemsets localement fréquents et sa frontière négative respectivement sur l'entrepôt de données W_i du site S_i. Une fois calculé, l'ensemble L_i^L sera transmet à l'« Agent Calcul_Itemsets_Globalement_Fréquents » (ou encore Agent CIGF) qui s'occupe de la tâche de génération des itemsets globalement fréquents sur la totalité de l'entrepôt de données.

Comme elle montre la figure ci-dessous le principe de fonctionnement de l'« Agent Calcul_Itemsets_Localement_Fréquents » peut être résumé en étapes suivantes :

- *Initialisation* : Dans cette étape, l'ensemble des itemsets localement fréquents L_i^L et sa frontière négative $FN(L_i^L)$ sont initialisés ($L_i^L = \phi$ et $FN(L_i^L) = \phi$).
- *Réception* : Dans cette étape l'« Agent Calcul_Itemsets_Localement_Fréquents » se bloque en attente de réception de tous les ensembles L_i^j et $FN(L_i^j)$, calculés par les « Agents Calcul_Itemsets_Fréquents » de tous les processeurs P_i^j de même site S_i.
- *Union* : À chaque réception de L_i^j et $FN(L_i^j)$, une étape d'union est exécutée afin de les combiner avec les autres ensembles d'itemsets déjà reçus.

$$\text{À chaque réception de } L_i^j \text{ et } FN(L_i^j) \text{ faire } \begin{cases} L_i^L = L_i^L \cup L_i^j \\ FN(L_i^L) = FN(L_i^L) \cup FN(L_i^j) \end{cases}$$

- *Traitement* : Une fois les ensembles L_i^j et $FN(L_i^j)$ de tous les processeurs P_i^j sont reçus, l' « Agent Calcul_Itemsets_Localement_Fréquents » génère l'ensemble L_i^L des itemsets localement fréquents et sa frontière négative $FN(L_i^L)$ comme suit :

$$L_i^L = \{x \;\; x \in (L_i^L \vee FN(L_i^L) \wedge x.count \geq minsup \times M_i)\}$$
$$FN(L_i^L) = \{x \;\; x \in (L_i^L \vee FN(L_i^L) \wedge x.coun \leq minsup \times M_i)\}$$

Où M_i est la taille de fragment W_i de l'entrepôt de données W.

- *Transmission* : Dans cette étape, l' « Agent Calcul_Itemsets_Localement_Fréquents » se bloque en attente de l'arrivée de l'« Agent Calcul_Itemsets_Globalement_Fréquents » au site S_i. Une fois arrivée, l'« Agent Calcul_Itemsets_Globalement_Fréquents » récupère l'ensemble L_i^L des itemsets

localement fréquents sur le site S_i (ou sur le fragment W_i de l'entrepôt de données W) par une communication avec l' « Agent Calcul_Itemsets_Localement_Fréquents ».

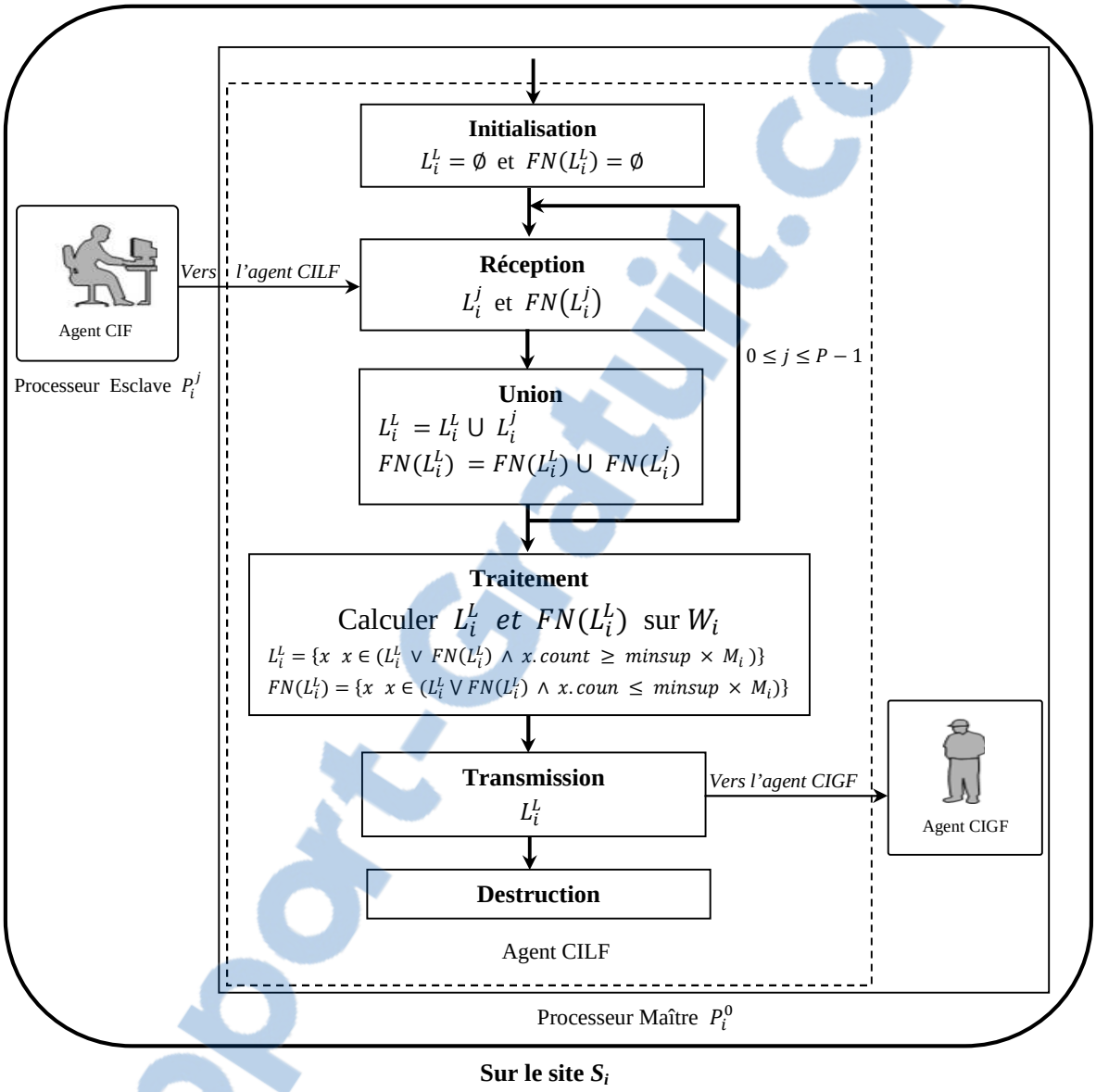


Figure 6. 22 Principe de fonctionnement de l'« Agent Calcul_Itemsets_Localement_Fréquents »

L'algorithme ci-dessous décrit le code de l'« Agent Calcul_Itemsets_Fréquents ».

Algorithme 6. 4 Code « Agent Calcul_Itemsets_Localement_Fréquents »

Début

$L_i^L = \phi$;

$FN(L_i^L) = \phi$;

Pour toutes $0 \leq j \leq (P-1)$ **Faire**

Réception des ensembles L_i^j et $FN(L_i^j)$;

$L_i^L = L_i^L \cup L_i^j$;

$FN(L_i^L) = FN(L_i^L) \cup FN(L_i^j)$;

Fin Pour ;

Pour tous itemsets $x \in L_i^L$ **Faire**

Si x.Count < $minsup \times M_i$ **Alors**

$L_i^L = L_i^L - \{ x \}$;

$FN(L_i^L) = FN(L_i^L) \cup \{ x \}$;

Fin si ;

Fin Pour ;

Pour tous itemsets $x \in FN(L_i^L)$ **Faire**

Si x.Count $\geq minsup \times M_i$ **Alors**

$FN(L_i^L) = FN(L_i^L) - \{ x \}$;

$L_i^L = L_i^L \cup \{ x \}$;

Fin si ;

Fin Pour ;

Transmettre l'ensemble L_i^L à l'agent « Calcul_Itemsets_Globalement_Fréquents » ;

Fin.

6. 8. 2 Génération des itemsets globalement fréquents

Pour générer l'ensemble des itemsets globalement fréquents, le site superviseur envoi un agent mobile, appelé « Agent Calcul_Itemsets_Globalement_Fréquents » (ou encore « Agent CIGF »), à tous les sites S_i du réseau afin de récupérer les itemsets localement fréquents sur chaque fragment W_i de l'entrepôt de données W. Donc, comme elle illustre la figure 6. 23, l'« Agent Calcul_Itemsets_Globalement_Fréquents » se déplace d'un site à un autre, ou plus précisément d'un processeur maître à un autre, et sur chaque site S_i (ou sur chaque processeur maître P_i^j), il récupère par une communication avec l'« Agent Calcul_Itemsets_Localement_Fréquents » l'ensemble L_i^L des itemsets localement fréquents sur le fragment W_i de l'entrepôt de données W. Au retour au site superviseur, l'« Agent Calcul_Itemsets_Globalement_Fréquents » génère l'ensemble L^G des itemsets globalement fréquents comme étant l'ensemble des itemsets qu'ont un support dépasse le seuil minsup spécifié par les utilisateurs.

$$L^G = \{x \;\; x \in L^G \wedge x.Count \leq minsup \times \sum_i^n M_i\}$$

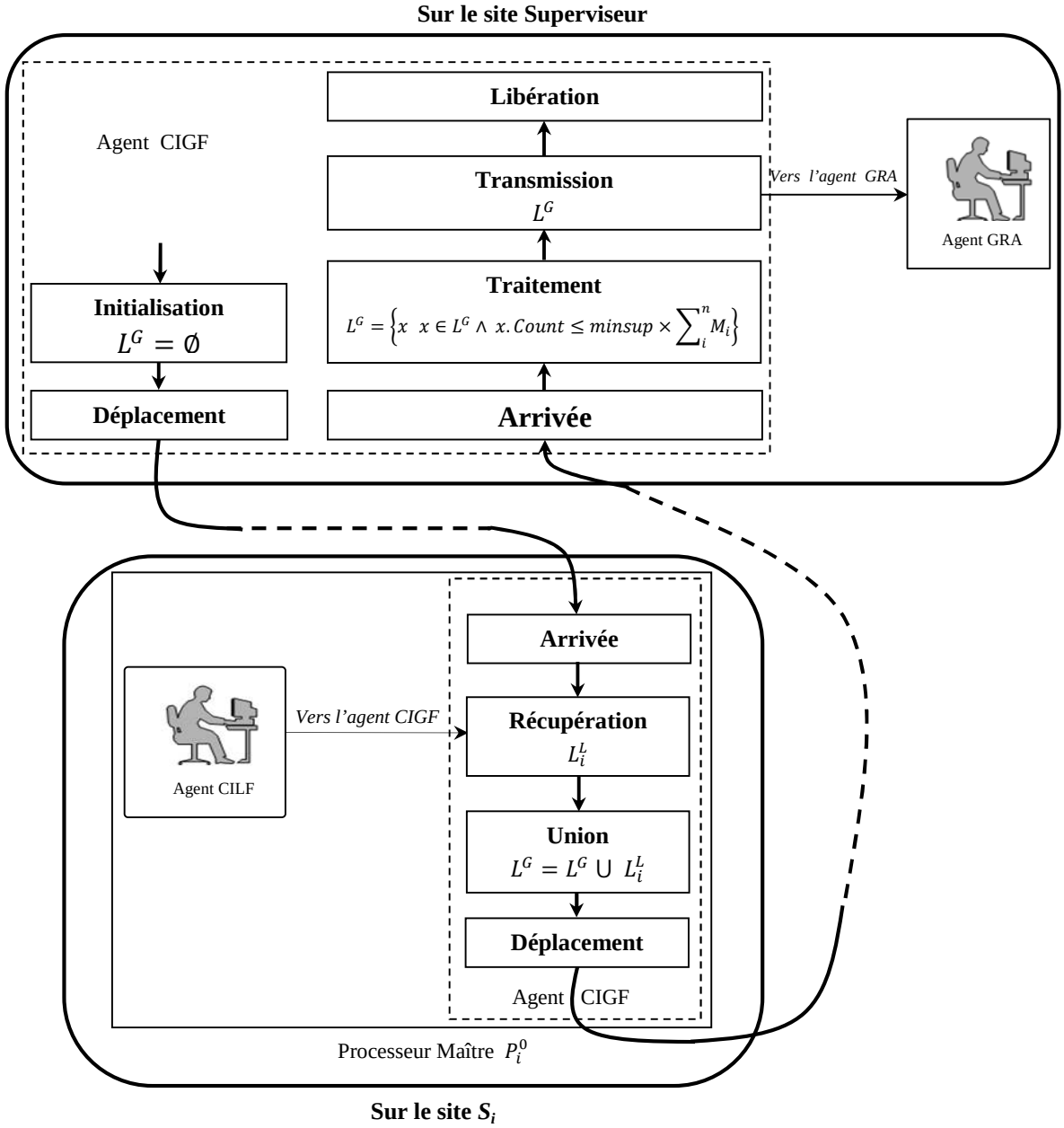


Figure 6. 23 Principe de fonctionnement de l'« Agent Calcul_Itemsets_Globalement_Fréquents »

Le code de l'« Agent Calcul_Itemsets_Globalement_Fréquents » peut être décrit comme suit :

Algorithme 6. 5 Code « Agent Calcul_Itemsets_Globalement_Fréquents »

Début

$L^G = \phi$;

Pour toutes $1 \leq i \leq n$ **Faire**

Récupérer l'ensemble L_i^L ;

$L^G = L^G \cup L_i^L$;

Fin Pour ;

Pour tous itemsets $x \in L^G$ **Faire**

Si x.Count < $minsup \times \sum_{i=1}^{n} M_i$ **Alors**

$L^G = L^G - \{ x \}$;

Fin si ;

Fin Pour ;

Transmettre l'ensemble L^G à l'« Agent Générer_Règles_Association » ;

Fin.

6. 8. 3 Génération des règles d'association

La génération de règles d'association s'effectue par l'exécution d'un agent stationnaire, appelé « Agent Générer_Règles_Association » (ou encore « Agent GRA »). Cet agent s'exécute en étapes suivantes (voir la figure 6. 24) :

- *Initialisation* : Dans cette étape, l'ensemble de règles d'association R est initialisé ($R = \phi$).
- *Réception* : Dans cette étape, l'« Agent Générer_Règles_Association » se bloque en attente de la réception l'ensemble L^G des itemsets globalement fréquents.
- *Traitement* : Une fois l'ensemble L^G est reçu, l'« Agent Générer_Règles_Association » génère l'ensemble de règles d'association comme suit :

$$\mathrm{R} = R \cup \left\{ \mathrm{x} \Longrightarrow (\mathrm{l} - \mathrm{x}) \text{ tel que } \mathrm{x} \subset \mathrm{l}, \mathrm{x} \neq \emptyset, \mathrm{l} \neq \emptyset, \frac{\mathrm{l.Count}}{\mathrm{x.Count}} \geq \mathrm{minconf} \right\}$$

- *Affichage* : La visualisation de l'ensemble de règles d'association sur l'écran s'effectue dans cette étape.

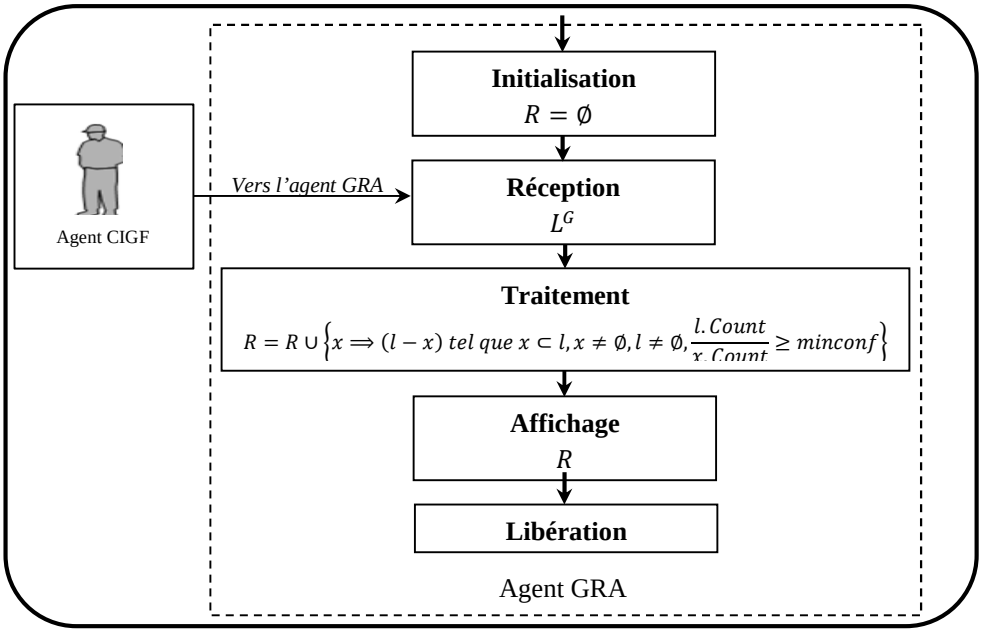


Figure 6. 24 Principe de fonctionnement de l'« Agent Générer_Règles_Association »

Le code de l'« Agent Générer_Règles_Association » peut être décrit comme suit :

Algorithme 6. 6 Code « Agent Générer_Règles_Association »

Début

Récupérer l'ensemble L^G ;

$R = \phi$;

Pour tous itemset $l \in L^G$ **Faire**
Pour tous sous itemset $x \subset l \ \wedge \ x \neq \phi \ \wedge \ x \neq l$ **Faire**
Si $\frac{l.Count}{x.Count} \geq minconf$ **Alors**
$R = R \cup \{ x \ \Rightarrow (l - x)\}$;
Fin Si ;
Fin Pour ;
Fin Pour ;
Fin.

6. 8. 4 En cas de mise à jour de l'entrepôt de données

Dans cette section, nous détaillons le principe de fonctionnement de l'approche proposée en cas de mise à jour de l'entrepôt de données.

6. 8. 4. 1 Mise à jour des itemsets localement fréquents

Sur chaque site S_i du réseau, l'ensemble w^+ de transactions ajoutées à l'entrepôt de données W_i est distribué équitablement entre les différents processeurs P_i^j de ce site, et chaque processeur exécute l'« Agent MAJ_Itemsets_Fréquents » qui s'occupe de la tâche de mise à jour de l'ensemble des itemsets fréquents et sa frontière négative sur son espace de données. Le principe de fonctionnement de cet agent peut être résumé en deux étapes suivantes (voir la figure 6. 25) :

- *Mise à jour* : Dans cette étape, les ensembles L_i^j et $FN(L_i^j)$ des itemsets fréquents et sa frontière négative respectivement sont mise à jour sur l'entrepôt de données rafraichi, W_i^+. Une fonction appelée « MAJ_Itemsets_Fréquents » (ou encore « fonction MAJ_IF »), qui porte l'algorithme « Mise_à_jour_Itemsets_Fréquents » décrit dans la section 6. 7, est utilisée pour réaliser cette tâche.
- *Transmission* : Dans cette étape, les ensemble L_i^j et $FN(L_i^j)$ mises à jour sur l'entrepôt de données W_i^+ du site S_i sont envoies à l'« Agent Mise_à_jour_Itemsets_Localement_Fréquents ».

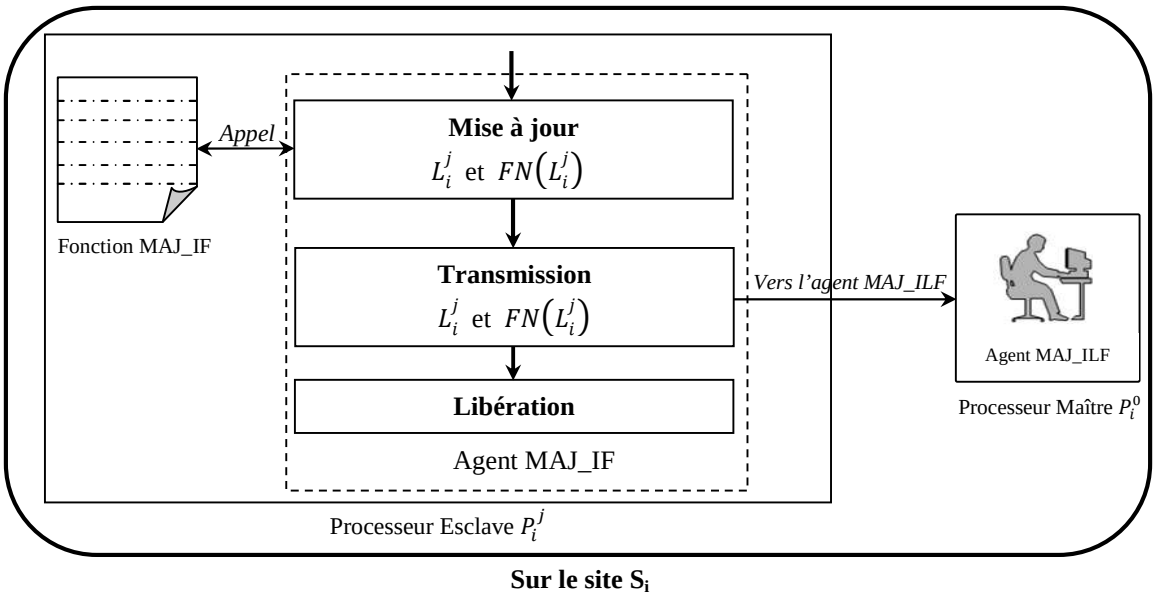


Figure 6. 25 Principe de fonctionnement de l'« Agent MAJ_Itemsets_Fréquents »

L'algorithme ci-dessous décrit le Code de l'« Agent MAJ_ Itemsets_Fréquents ».

Algorithme 6. 7 Code « Agent MAJ_ Itemsets_Fréquents »

Début

Mise_à_jour_Itemsets_Fréquents (L_i^j , $FN(L_i^j)$) ;

Transmettre L_i^{j+} et $FN(L_i^{j+})$ à l' « Agent MAJ_Itemsets_Localement_Fréquents »

Fin.

Sur chaque processeur maître P_i^0 de chaque site S_i, l'« Agent MAJ_Itemsets_Localement_Fréquents » attend la réception de tous les ensembles L_i^{j+} et $FN(L_i^{j+})$, mises à jour par les « Agents MAJ_Itemsets_Fréquents » de tous les processeurs P_i^j de même site S_i, et les combines ensuite en ensembles L_i^{L+} et $FN(L_i^{L+})$ des itemsets localement fréquents et sa frontière négative respectivement sur l'entrepôt de données W_i^+. Une fois calculé, l'ensemble L_i^{L+} sera transmet à l'« Agent MAJ_Itemsets_Globalement_Fréquents » (ou encore Agent MAJ_IGF) qui s'occupe de la tâche de mise à jour des itemsets globalement fréquents sur l'entrepôt de données rafraichi, W^+. Le principe de fonctionnement de cet agent est décrit par l'algorithme suivant :

Algorithme 6. 8 Code « Agent MAJ_Itemsets_Localement_Fréquents »

Début

$L_i^{L+} = \phi$;

$FN(L_i^{L+}) = \phi$;

Pour toutes $0 \leq j \leq (P-1)$ **Faire**

Réception des ensembles L_i^{j+} et $FN(L_i^{j+})$;

$L_i^{L+} = L_i^{L+} \cup L_i^{j+}$;

$FN(L_i^{L+}) = FN(L_i^{L+}) \cup FN(L_i^{j+})$;

Fin Pour ;

Pour tous itemsets $x \in L_i^{L+}$ **Faire**

Si x.Count < $minsup \times M_i^+$ **Alors**

$L_i^{L+} = L_i^{L+} - \{ x \}$;

$FN(L_i^{L+}) = FN(L_i^{L+}) \cup \{ x \}$;

Fin si ;

Fin Pour ;

Pour tous itemsets $x \in FN(L_i^{L+})$ **Faire**

Si x.Count $\geq minsup \times M_i^+$ **Alors**

$FN(L_i^{L+}) = FN(L_i^{L+}) - \{ x \}$;

$L_i^{L+} = L_i^{L+} \cup \{ x \}$;

Fin si ;

Fin Pour ;

Lancer l'« Agent MAJ_Itemsets_Globalement_Fréquents » ;

Fin.

6. 8. 4. 2 Mise à jour des itemsets globalement fréquents

Une fois l'ensemble des itemsets fréquents et sa frontière négative sont met à jour sur l'entrepôt de données rafraîchi, l'« Agent MAJ_Itemsets_Localement_Fréquents » lance un agent mobile, appelé « Agent MAJ_Itemsets_Globalement_Fréquents » (ou encore « Agent MAJ_IGF »), qui s'occupe de la mise à jour de l'ensemble des itemsets globalement fréquents. Cet agent déplace d'un site à un autre, démarrant de site de l'entrepôt de données rafraîchi, pour récupérer les ensembles des itemsets localement fréquents des autres entrepôts de données (ou des autres fragments de l'entrepôt de données global). Une fois tous les sites du réseau sont visités, l' « Agent MAJ_Itemsets_Globalement_Fréquents » génère l'ensemble des itemsets globalement fréquents mis à jour et enfin, il déplace au site superviseur pour lancer l'« Agent Générer_Règles_Association » qui s'occupe de la génération de l'ensemble de règles d'association mis à jour. L'algorithme suivant décrit le code de l'« Agent MAJ_Itemsets_Globalement_Fréquents ».

Algorithme 6. 9 Code « Agent MAJ_Itemsets_Globalement_Fréquents »

Début

$L^{G+} = L_i^{L+}$;

Pour toutes $1 \le k \le n$ et $k \ne i$ **Faire**

Récupérer l'ensemble L_k^L ;

$L^{G+} = L^{G+} \cup L_k^L$;

Fin Pour ;

Pour tous itemsets $x \in L^{G+}$ **Faire**

Si x.Count < $minsup \times \sum_{i=1}^{n} M_i^+$ **Alors**

$L^{G+} = L^{G+} - \{ x \}$;

Fin si ;

Fin Pour ;

// Sur le site superviseur

Lancer l'« Agent Générer_Règles_Association » ;

Fin.

6. 9 Conclusion

Dans ce chapitre, nous avons abordés, dans un premier temps, la problématique d'extraction de règles d'association à partir des grandes bases de données, telles que les entrepôts de données, en proposant des améliorations pour deux algorithmes très utilisés pour l'extraction de règles d'association à partir des grandes masses de données, l'algorithme Partition et l'algorithme Incremental Mining. Par la suite, nous avons essayés de résoudre la problématique d'extraction de connaissances à partir d'un entrepôt de données distribué, en proposant une approche parallèle et distribuée d'extraction de connaissances exprimables sous forme de règles d'association basée sur des agents mobiles et stationnaires. Dans cette approche, la génération des itemsets localement fréquents sur chaque site du réseau (ou sur chaque fragment de l'entrepôt de données global) et leurs mise à jour en cas de rafraîchissement de l'entrepôt de données exploite les algorithmes proposées, l'algorithme Partition amélioré et l'algorithme Incremental Mining respectivement

Conclusion et perspectives

Dans ce travail, nous nous sommes intéressés à la problématique d'extraction des connaissances à partir des entrepôts de données distribués. Dans cette problématique, nous nous focalisant particulièrement sur les connaissances exprimable sous forme de règles d'association. Avant de présenter nos contributions, une étude détaillée sur les différents concepts qui intervient à cette problématique est réalisée. Ces concepts peuvent être résumés en Extraction de Connaissances à partir de Données (ECD), Entrepôt de Données et l'Extraction Parallèle et Distribution des Connaissances (EPDC).

Notre première contribution concerne la proposition d'un algorithme pour la génération des itemsets fréquents. Comme les entrepôts de données sont des bases de données volumineuses, l'algorithme Partition qui s'exécute en deux passes seulement est souvent considéré le plus approprié pour l'extraction de règles d'association. On ce basant sur ce point, nous avons proposés une amélioration pour cet algorithme avant de l'appliqué dans notre approche. L'idée de base est d'exécuter l'algorithme Partition d'une façon incrémentale et à chaque traitement d'une nouvelle partition, l'algorithme mis à jour ces connaissances (itemsets fréquents, itemsets non fréquents et itemsets candidats) sur les données traitées. Ainsi, cette démarche incrémentale nous a permet de porter un gain de performances en exploitant au maximum les calculs effectués durant la première phase de l'algorithme.

Ensuite, notre deuxième contribution considère le cas de mise à jour de l'entrepôt de données en proposant un algorithme de mise à jour des itemsets fréquents. Ce dernier présente une amélioration pour l'algorithme Incremental Mining, qui tente de pousser encore mieux l'étape de parcourt complet de l'entrepôt de données.

Enfin, nous présentons une approche parallèle et distribuée d'extraction de connaissances exprimables sous forme de règles d'association à partir d'un entrepôt de données distribué sur les différents sites du réseau. Cette approche est basée sur un ensemble d'agents mobiles et stationnaires qui coopèrent pour extraire à la fois des connaissances locales (itemsets localement féquents) et globales (itemsets globalement fréquents et règles d'association). L'aventage de cette approche n'est pas seulement elle exploiter les concepts d'agents mobiles qui apporte des solutions efficaces aux problèmes distribués, elle exploite également le traitement parallèle sur chaque nœud du réseau ; ce qui améliore les performances de notre approche.

Les perspectives pour notre travail sont très nombreuses :

- Nous souhaitons implémenter notre approche sur un réseau de grande échelle tel que les grilles de calcul.
- Nous pouvons améliorer notre approche en exploitant une communication hiérarchique entre les agents stationnaires d'un même site ou, en partitionnant les sites en groupes et chaque groupe d'agents s'occupe d'un sous groupe de sites.

Bibliographie

[Buc 2006] B.G. Buchanan. « Brief History of Artificial Intelligence ». Retrieved March 22, 2006.

[Kod 1967] Y. Kodratoff. « Technical and Scientific Issues of KDD ». Algorithms Learning Theory, 997:261-297, 1967.

[Pia 1991] G. Piatetsky-Shapiro. « Knowledge Discovery in Real Databases », A Report on the IJCAI-89 Workshop. AI Magazine 11(5): 68–70, 1991.

[Fay 1996a] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth. « From Data Mining to Knowledge Discovery in Databases ». AI Magazine, 17: 37-54, 1996.

[Han 2000] J. Han, M. Kamber. « Data Mining: Concepts and Techniques », Morgan Kaufmann Publishers, 2000.

[Fay 1996b] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth. « Knowledge Discovery and Data Mining: Towards a Unifying Framework ». AI Magazine, 37-54, 1996.

[Bra 1996] R. Brachman, T. Anand. « The Process of Knowledge Discovery in Databases: A Human-Centered Approach ». In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy, (eds.), Adv. in KDDM, MIT Press, Cambridge, 1996.

[Sil 1995] A. Silberschatz, A. Tuzhilin. « On Subjective Measures of Interestingness in Knowledge Discovery ». In proceedings of KDD-95. First International Conference on KDDM, PP. 275-281, Monlo Park, CA: AAAI Press, 1995.

[Gar 2000] G. Gardrin. « Internet / Intranet et Bases de Données: Data Web, Data Media, Data Warehouse - Data Mining », Eyrolles, 2000.

[Nai 1986] J. Naisbitt, Megatrends, 6th ed., Warner Books, New York, 1986.

[Fu 2001] Y. Fu. « Distributed Data Mining : An Overview ». Newsletter of the IEEE Technical Committee on distributed Processing, Spring 2001, pp. 5-9, 2001.

[Par 2003] B. Park, H. Kargupta. « Distributed Data Mining: Algorithms, Systems, and Applications ». In N. Ye (Ed.), « The Handbook of Data Mining ». (pp. 341-358). Lawrence Erlbaum Associates, 2003.

[Guo 1999] Y. Guo, R. Grossman. « Scalable Parallel and Distributed Data Mining ». Data Mining and Knowledge Discovery, 3, Sept, 1999.

[Zak 2000] M. J. Zaki, « Parallel and Distributed Data Mining: An Introduction ». In Mohammed J. Zaki, Ching-Tien Ho (eds.), « Large-Scale Parallel Data Mining ». Springer-Verlag, 2000.

[Fay 1996c] U. Fayyad. « Data Mining and Knowledge Discovery: Making Sense Out of Data ». IEEE Expert, 5. 11, no. 5, pp. 20-25, October 1996.

[Fay 1996d] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth. « From Data Mining to Knowledge Discovery: An Overview ». In U. Fayyad, G. P. Shapiro, Amith, P. Smyth, R. Uthurusamy, (eds.), Adv. in KDDM, MIT Press, 1-36, Cambridge, 1996.

[Fay 1996e] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth. « The KDD Process for Extracting Useful Knowledge from Volumes of Data ». Communications of the ACM, 5. 39, no. 11, pp. 27-34, November 1996.

[Fam 1997] A. Famili, W. M. Shen, R. Weber, E. Simoudis. « Data Preprocessing and Intelligent Data Analysis ». Intelligent Data Analysis, Vol. 1, P. 3-23, 1997.

[Cal 2001] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi et R. Rosat1. « Data Integration in Data Warehousing ». Int. Journal. Of Cooperative Information Systems, Vol. 10, P. 237-271, 2001.

[Pyl 1999] D. Pyle. « Data Preparation for Data Mining ». Morgan Kaufmann Publishers, 1999.

[Gil 2002] R. Gilleron, M. Tommas1. « Découverte de Connaissances à partir de Données. Notes de cours », IUP MIAGE, Lille, Octobre 2002.

[Tuf 2003] S. Tufferry. « Data Mining et Scoring - Gestion de la Relation Client », Eyrolles, Paris 2003.

[Lef 1998] R. Lefébure, G. Ventur1. « Le Data Mining ». Eyrolles, Paris, 1998.

[Lar 2005] D. T. Larose. « Discovering Knowledge in Data : An Introduction to Data Mining ». John Wiley & Sons Publishers, Canada, 2005.

[Che 1996a] M. Chen, J. Han, P. S. Yu. « Data Mining : An Overview from Database Perspective ». IEEE Transactions on KDE, 8 (6), 866-883, 1996.

[Pas 1998] D. Pascot. « Mise en contexte : l'entrepôt de données clé de voûte de l'informatique décisionnelle ». Technical report, Université de Laval, 1998.

[Bou 2003] O. Boussaid, F. Bentayeb, J. Darmont, and S. Rabaséda. « Vers l'entreposage des données complexes : Structuration, intégration et analyse ». Ingénierie des Systèmes d'Information, 8(5-6) : 79-107, 2003.

[Wie 1992] G. Wiederhold. « Mediators in the Architecture of Future Information Systems ». IEEE Computer, 25(3): 38-49, Mars 1992.

[Wid 1995] J. Widom. « Research Problems in Data Warehousing ». In CIKM'95, pages 25-30, New York, NY, 1995. ACM Press.

[Inm 1996] W. H. Inmon. « Building the Data Warehouse », John Wiley, N. York, 2nd edition, 1996.

[Cle 2001] C. Clemmen. « Amélioration de la qualité des données dans les entrepôts de données et son impact dans les pratiques organisationnelles : Application aux bases de données administratives de l'ULB ». 2ème Licence en informatique, Mons-Hainaut, 2001.

[Mar 2006] F. Marhoumi. « Entrepôts de données XML : Développement d'un outil Extraction Transformation Load ETL ». Ing. Civil Informaticien en Sc. App., Libre de Bruxelles, 2006.

[Bal 1998] C. Ballard, D. Herreman, D. Schau, R. Bell, E. Kim, A. Valencic. « Data Modeling Techniques for Data Warehousing ». Int. Technical Support Organization, IBM, 1998.

[Cha 1997] S. Chaudhuri, U. Dayal. « An Overview of Data Warehousing and OLAP Technology », SIGMOD Record, 26(1): 65-74, 1997.

[Kim 2000] R. Kimball, L. Reeves, M. Ross, W. Thornthwaite. « Concevoir et Déployer un Data Warehouse », Eyrolles, Octobre 2000.

[Tri 2005] M. Trinidad S. Encinas. « Entrepôt de données pour l'aide à la décision médicale: conception et expérimentation ». Thèse de doctorat, J. Fourier, 2005.

[Inm 1993] W. H. Inmon. « The Operational Data Store ». PRISM Tech Topic, Vol. 1, No. 17, 1993.

[Sou 2005] A. Soussi, J. Feki, F. Gargouri. « Approche semi-automatisée de conception de schémas multidimensionnels valides ». (EDA 05), Lyon, vol. B-1 of Revue des Nouvelles Technologies de l'Informatique, pp 71-90, Cépaduès Editions, 2005.

[Phi 2002] C. Philipps, K. C. Davis. « Automating Data Warehouse conceptual Schema Design and Evaluation ». In IVth Int. Workshop on DMDW, Toronto, Canada, vol. 58 of CEUR Workshop Proc., pp 23-32. CEURWR. Org, 2002.

[Kim 2003] H. J. Kim, T. H. Lee, S. G. Lee, J. Chun. « Automated Data Warehousing for Rule-Based CRM Systems ». In XIVth Australasian Database Conference on database Technologies, pp 67-73. Australian Computer Society, 2003.

[Try 1999] N. Tryfona, F. Busborg, J. G. Borch Christiansen. « starER : A Conceptual Model for Data Warehousing Design ». In DOLAP 99, Missouri, USA, pp 3-8. ACM Press, 1999.

[Luj 2005] S. Luján-Mora. « Data Warehouse Design with UML ». PhD Thesis, Alicante, Juin 2005.

[Gol 1998] M. Golfarelli, D. Maio, S. Rizz1. « The Dimensional Fact Model : A conceptual Model for Data Warehouses ». Int. Journal of Cooperative Inf. Syst., 7(2-3) : 215-247, 1998.

[Fav 2007] C. Favre. « Évolution de schémas dans l'entrepôt de données : Mise à jour de hiérarchies de dimension pour la personnalisation des analyses ». Thèse Doctorat, Lyon 2, 2007.

[Bon 2001] A. Bonifati, F. Cattaneo, S. Ceri, A. Fuggetta, S. Parabosch1. « Designing Data Marts for Data Warehouses ». ACM Transactions on Soft. Eng. and Meth., 10(4): 452-483, 2001.

[The 2000] D. Theodoratos, M. Bouzeghoub. « A general framework for the view selection problem for Data Warehouse design and evolution ». DOLAP 00, ACM 3rd Int. Workshop on Data Warehousing and OLAP, 2000.

[Gut 2000] A. Gutiérrez, A. Marotta « An Overview of Data Warehouse Design Approaches and Techniques », Instituto de Computación, 2000.

[Agr 1995a] R. Agrawal, A. Gupta, S. Sarawag1. « Modeling Multidimensional Databases ». IBM Research Report, 1995.

[Agr 1997] R. Agrawal, A. Gupta, A. Sarawagi. « Modeling Multidimensional Databases ». ICDE'97, 1997.

[Li 1996] C. Li, X. S. Wang. « A Data Model for Supporting On-Line Analytical Processing ». In Proceedings of the 5th Int. Conf. on Information and Knowledge Management, 1996.

[Gys 1997] M. Gyssen, L. 5. S. Lakshmanan. « A Foundation for Multi-Dimensional Databases ». In Proceedings of 23rd International Conference on VLDB, Athens, August 25-29, 1997.

[Gup 1995] A. Gupta. I. S. Mumick. « Maintenance of Materialized Views : Problems, Techniques, and Applications ». IEEE Data Engineering Bulletin, 1995.

[Yan 1997] J. Yang, K. Karlapalem, Q. Li. « Algorithms for materialized view design in data warehousing environment ». Proc. VLDB'97, 1997.

[The 1999] D. Theodoratos, T. Sellis. « Designing Data Warehouses ». DKE 31, (3) : 279-301, 1999.

[Vow 2002] J. Vowler. « Data Warehouse design from top to bottom ». ComputerWeekly. 2002.

[Eck 2002] W. Eckerson. « Four ways to build a data warehouse ». App. development trends, 2002.

[Kim 1996] R. Kimball. « Entrepôts de données, Guide pratique du concepteur de Data Warehouse ». John Wiley and Sons, Inc., 1996.

[Mar 1998] P. Marcel. « Manipulation de données multidimensionnelles et langages de règles ». PhD thesis, Inst. Nat. des Sciences Appliques de Lyon, France, 1998.

[Ben 2006a] R. Ben Messaoud. « Couplage de l'analyse en ligne et de la fouille de données pour l'exploration, l'agrégation et l'explication des données complexes ».Thèse de Doctorat, Lumière Lyon 2, 2006.

[Cab 1998] L. Cabibbo, R. Torlone. « A Logical Approach to Multidimensional Databases ». EDBT'98, Valencia (Spain), March 1998.

[Moh 1999] M. Mohania, S. Samtani, J. Roddick, Y. Kambayashi. « Advances and Research Directions in Data Warehousing Technology ». AJIS, Volume 7, Number 1, 1999.

[Tes 2000] O. Teste. « Modélisation et manipulation d'entrepôts de données complexes et historisées ». PhD thesis, Paul Sabatier de Toulouse, 2000.

[Pre 2005] Ph. Preux. « Fouille de données ». Notes de cours, Lille 3, 2005.

[An 2006] A. An. « Classification Methods ». York University, Canada. In J. Wang. « Encyclopedia of Data Warehousing and Mining ». Montclair State University, USA, Idea Group Ref., 2006.

[Gri 2006] L. Grivel. « Outils de classification et de catégorisation pour la fouille de textes ». Equipe ISIS, Université de Marne-La-Vallée, 2006.

[Qui 1986] J. R. Quinlan. « Induction of Decision Trees ». Mach. Lear., 1(1):81-106, 1986.

[Qui 1993] J. R. Quinlan. « C 4.5: Programs for machine learning ». M. Kaufmann, San Mateo, 1993.

[Bre 1984] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone. « Classification And Regression Trees ». Wadsworth International Group, Belmont, CA, 1984.

[Har 1975] J. A. Hartigan. « Clustering Algorithms ». J. Wiley, New York, 1975.

[Sha 1948] C. Shannon. « A mathamatical theory of communication ». The Bell System Technical Journal, 27, 1948.

[Leb 1998] L. Lebart, A. Morineau, and M. Piron. « Statistiques Exploratoire Multidimensionnelle ». Paris, Dunod edition, 1998.

[Cor 2003] A. Cornuéjols and L. Miclet. « Apprentissage artificiel : Concepts et algorithmes ». Paris, Eyrolles edition, 2003.

[Pea 1988] J. Pearl. « Probabilistic reasoning in intelligent systems : networks of plausible inference ». Morgan Kaufmann, 1988.

[Fri 1996] N. Friedman M. Goldszmidt. « Building classifiers using bayesian networks ». Proc. of the American association for art. intellig. conf. (AAAI'96), 1996.

[Ben 2006b] N. Ben Amor, S. Benferhat, Z. Elouedi. « Réseaux bayésiens naïfs et arbres de décision dans les systèmes de détection d'intrusions ». RSTI-TSI, V. 25, N. 2/2006, P. 169-196, 2006.

[Mac 1943] W. S. MacCullogh, W. Pitts. « A logical Calculus of the Ideas Immanent in Nervous Activity ». Bull. Of Math. Biophysics 5, 1943.

[Dav 1993] E. Davalo, P. Naïm. « Des réseaux de neurones ». Eyrolles, Paris, 1993.

[Ros 1958] F. Rosenblatt. « The Perceptron : A probabilistic model for information storage and organization in the brain ». Psychologie Review, 1965.

[Dre 2002] G. Dreyfus. « Réseaux de neurones : Méthodologie et applications ». Eyrolles, 2002.

[Min 1969] M. Minsky, S. Papert. « Perceptron ». Mit Press, 1969.

[Jod 1992] J. F. Jodouin. « Les réseaux neuromimétiques (modèles et applications) ». Hermes, 1992.

[Rum 1986] D. E. Rumelhart, G. E. Hinton, R. J. Williams. « Learning Internal representations by Error Propagation ». PDP, MIT Press, 1986.

[Ben 1973] J. P. Benzécr1. « L'analyse de données ». Dunod, Paris, 1973.

[Cai 1976] F. Caillez, J. P. Pages. « Introduction à l'analyse de données ». S.M.A.S.H, Paris, 1976.

[Rou 1986] M. Roux. « Algorithmes de classification ». Masson, Paris, 1986.

[Cel 1989] G. Celeux, E. Diday, G. Govaert, Y. Lechevallier, and H. Ralambondrainy. « Classification automatique des données – environnement statistique et informatique ». Dunod, Paris, 1989.

[Jai 1988] A. K. Jain, R. Dubes. « Algorithms for Clustering Data ». Prentice Hall, New Jersey, 1988.

[Jai 1999] A. K. Jain, M. N. Murty, and P. J. Flynn. « Data Clustering: A review ». ACM Computing Surveys, 31(3): 264-323, 1999.

[Ber 2002] P. Berkhin. « Survey of Clustering Data Mining Techniques ». Technical Report, accrue software, San Jose, CA, 2002.

[For 1965] E. W. Forgy. « Cluster analysis of multivariate data: efficiency versus interpretability of classification ». Biometrics, vol. 21(3), p. 768, 1965.

[Mac 1967] J. B. MacQueen. « Some methods for classification and analysis of multivariate Observations ». Proc. Symp. Math. Stat. And probability (5th), Un4. Of California, Berkeley, p 281-297, 1967.

[Did 1971] E. Diday. « La méthode des nuées dynamiques ». Revue statistique Appliquée, vol. 19(2), p. 19-34, 1971.

[Hua 1997a] Z. Huang. « A fast clustering algorithm to cluster very large categorical data sets in data mining ». SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Tucson, Arizona, 1997.

[Hua 1997b] Z. Huang. « Clustering Large Data Sets with Mixed Numeric and Categorical Values ». Proc. of The 1st Pacific-Asia Conf. on KDDM, Singapore, 1997.

[Kau 1990] L. Kaufman, and P. Rousseeuw. « Finding Groups in Data : An Introduction to Cluster Analysis », John Wiley and Sons, 1990.

[Ng 1994] R. T. Ng, J. Han. « Efficiency and effective clustering method for spatial data mining ». Int. Conf. VLDB, Santiago, Chile, p 144-155, 1994.

[Dem 1977] A. Dempster, N. Liard, D. B. Rubin. « Maximum likelihood from incomplete data via the EM algorithms ». J. Royal Statist. Soc. B, 39 :1-39, 1977.

[Guh 1998] S. Guha, R. Rastogi, and K. Shim. « CURE: an efficient clustering algorithm for large databases ». In ACM SIGMOD Int. Conf. on Management of Data, pages 73-84, 1998.

[Kar 1999] G. Karypis, E. Han, and 5. Kumar. « Chameleon: Hierarchical clustering using dynamic modelling ». IEEE Computer, 32(8): 68-75, 1999.

[Est 1996] M. Ester, H. P. Kriegel, J. Sander et X. Xu. « A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise ». In : 2nd Int. Conf. on KDDM, édité par E. Simoudis, J. Han, U. Fayyad. pp. 226-231, Portland, Oregon, 1996.

[Ank 1999] M. Ankerst, M. Breunig, H. P. Kriegel et J. Sander. « OPTICS : Ordering Points To Identify the Clustering Structure ». In : Proceeding of Int. Conf. Management of Data ACM-SIGMOD, PP. 49-60, Philadelphia, Pennsylvania, USA, 1999.

[Hin 1998] A. Hinneburg et D. A. Keim. « An Efficient Approach to Clustering in Large Multimedia Databases with Noise ». In : Knowledge Discovery and Data Mining, PP. 58-65, New York City, USA, 1998.

[Maz 2002] L. Mazlack, S. Coppock. « Using soft computing techniques to integrate multiple kinds of attributes in data mining ». In 5th Int. FLINS Conf., Computational Intellegent Systems for Applied Research, pp. 137-144, 2002.

[War 1963] J. H. Ward, « Hierarchical grouping to optimize an objective function ». Journal of the American Statistical Association, 58: 236-244, 1963.

[Agr 1993] R. Agrawal, T. Imielinski, A. Swami. « Mining association rules between sets of items in large databases ». In proceedings of the Int. Conf. on Management of Data (SIGMOD'93), pp. 207-216, Washington DC, USA, 1993.

[Agr 1994] R. Agrawal, R. Srikant. « Fast Algorithms for Mining Generalized Association Rules ». In Proceedings of the 20th International Conference on VLDB, Santiago, Chile, 1994.

[Man 1994a] H. Mannila, H. Toivonen, A. I. Verkano. « Improved Methods for Finding Association Rules ». Technical Report, University of Helsink1. Finland, 1994.

[Man 1994b] H. Mannila, H. Toivonen, A. I. Verkano. « Efficient algorithms for discovering association rules ». In KDD-94: AAAI Workshop on knowledge Discovery in Databases, pp. 181-192, Seattle, Washington, 1994.

[Hou 1995] M. Houtsma, A. Swami. « Set-Oriented Mining of Association Rules ». In Proceedings of International Conference on ICDE , Taipei, Taiwan, 1995.

[Sav 1995] A. Savasere, E. Omiecinski, S. Navthe. « An Efficient Algorithm for Mining Association Rules in Large Databases ». In Proc. of the 21st Int. Conf. on VLDB, p. 432-444, 1995.

[Sri 1995] R. Srikant R. Agrawal. « Mining Generalized Association Rules ». In Proceedings of the 21st International Conference on VLDB, Zurich, Switzerland, 1995.

[Soo 1995] J. Soo Park, M. Chen, P. Yu. « An effective hash based algorithm for mining association rules ». In proceedings of the International Conference on Management of Data (SIGMOD'95), San Jose, California, 1995.

[Bri 1997] S. Brin, R. Motwani, J. Ullman, S. Tsur. « Dynamic Itemset Counting and Implication Rules for Market Basket Data ». In Proc. of the ACM SIGMOD Conf., pp. 255-264, 1997.

[Bur 1998] J. C. Burges. « A tutorial on support vector machines for pattern recognition ». Data Mining and Knowledge Discovery, 2(2) :121-167, 1998.

[Agr 1995b] R. Agrawal and R. Srikant. « Mining Sequential Patterns ». In Proceedings of the 11th International Conference on ICDE, Taipei, Taiwan, 1995.

[Tso 1999] G. Tsoumakas, I. Vlahavas. « Distributed Data Mining ». Aristotle University of Thessaloniki, Greece. 1999.

[Tan 2008] D. Taniar, C. Leung, W. Rahayu, S. Goel. « High-Performance Parallel Database Processing and Grid Databases ». John Wiley, New Jersey, 2008.

[Fre 1998] A. Freitas, S. Lavington. « Mining Very Large Databases with Parallel Processing ». Kluwer Academic Publishers, ISBN 0-7923-8048-7, 1998.

[Agr 1996] R. Agrawal, J. C. Shafer. « Parallel mining of association rules ». IEEE Transactions on Knowledge and Data Eng., 8 (6): 962-969, 1996.

[Han 1997] E. Han, G. Karypis, V. Kumer. « Scalable parallel data mining for association rules ». In ACM SIGMOD International Conf. on Management of Data, 1997.

[Han 1996] E. Han, A. Srivastava, V. Kumar. « Parallel Formulations of Inductive Classification Learning Algorithm ». Technical Report TR-96-040, Department of Computer and information Sciences, Minnesota, 1996.

[Sto 1999] K. Stoffle, A. Belkonience. « Parallel k-means Clustering for Large Datasets». Proceedings of EuroPar-99, 1999.

[Jin 2001] R. Jin, G. Agrawal. « A Middleware for Developing Parallel Data Mining Implementations ». Proc. First SIAM Conf. Data Mining, April 2001.

[Kum 1994] V. Kumar, A. Grama, A. Gupta, G. Karypis. « Introduction to Parallel Computing : Algorithm Design and Analysis ». Benjamin Cummings, Redwod City, 1994.

[Sri 1997] A. Srivastava, V. Singh, E. H. Han, V. Kumar. « An Efficient, Scalable, Parallel Classifier for Data Mining ». Technical Report TR-97-010, Department of Computer Science, Minnesota, 1997.

[Sri 1999] A. Srivastava, E. H. Han, V. Kumar, V. Singh. « Parallel Formulations of Decision-Tree Classification Algorithms ». Data Mining and Knowledge Discovery : An International Journal, 3(3): 237-261, 1999.

[Ken 1997] Y. Kenji. « Distributed Cooperative Bayesian Learning Strategies ». In Proceedings of the 10th Annual Conference on Computational Learning Theory, ACM Press, Nashville, Tennessee, pp. 250-262, 1997.

[Won 1993] T. S. Wong. « Partitioning strategies for parallelizing neural networks ». Master's thesis, Queen's University, 1993.

[Rog 1997a] R. O. Rogers, D. B. Skillicorn. « Strategies for parallelizing supervised and unsupervised learning in artificial neural networks using the BSP cost model ». Technical Report TR-97-406, Queen's University, 1997.

[Rog 1997b] R. O. Rogers, D. B. Skillicorn. « A framework for parallel data mining using neural networks ». Technical Report TR-97-413, Queen's University, 1997.

[Dhi 2000] I. S. Dhillon, D. S. Modha. « A Data-Clustering Algorithm on Distributed Memory Multiprocessors ». Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence, 1759:245-260, 2000.

[Li 2006] J. Li, Y. Liu, W. Liao, A. Choudhary. « Parallel Data Mining Algorithms for Association Rules and Clustering ». CRC Press, LLC, 2006.

[Zak 1999] M. J. Zaki. « Parallel and Distributed Association Mining: A survey ». IEEE Conc., 1999.

[Shi 1996] T. Shintani, M. Kitsuregawa. « Hash based parallel algorithms for mining association rules ». In 4th Int. Conf. on Parallel and Distributed Inf. Sys., 1996.

[Par 1995] J. S. Park, M. Chen, P. S. Yu. « Efficient parallel data mining for association rules ». In ACM Int. Conf. on Information and Knowledge Management, 1995.

[Che 1996b] D. Cheung, J. Han, V. Ng, A. Fu, Y. Fu. « A Fast Distributed Algorithm for Mining Association Rules ». 4th Int. Conf. on Parallel and Distributed Information systems, 1996.

[Zak 1996] M. J. Zaki, M. Ogihara, S. Parthasarathy, W. Li. « Parallel Data Mining for Association Rules on Shared-Memory Multi-processors », Technical Report TR 618, Rochester, Computer Science, 1996.

[Val 2003] Hima Vallikona. « Association Rules Mining over Multiple Databases : Partitioned and Incremental Approaches ». Master of Science in Computer Science and Engineering, Texas At Arlington, 2003.

[Fer 1995] J. Ferber. « Les systèmes Multi-Agents : vers une intelligence collective ». InterEdition, 1995.

[Jen 2000] N. Jennings, M. Wooldridge. « Agent-Oriented Software Engineering », in Handbook of Technology (ed. J. Bradshaw), AAAI/MIT Press, 2000.

[Sek 2007] M. Sekma. « Intégration de protocoles de sécurité pour la communication inter-agents dans la plate-forme Aglets ». Inst. Sup. d'Informatique et de Mathématiques, Monastir, 2007.

[Zga 2007] H. Zgaya. « Conception et optimisation distribuée d'un système d'information d'aide à la mobilité urbaine ». Thèse Doctorat, École centrale de Lille, 2007.

[Cha 1999] B. Chaïb-Draa. « Agents et Systèmes Multi-Agents ». Notes de cours, Département informatique, Faculté des sciences et de génie, université Laval, Québec, 1999.

Annexe

Agents Mobiles et la Plateforme Aglets

I Introduction

L'approche proposée, dans le chapitre 6, pour l'extraction de règles d'association à partir d'un entrepôt de données distribué est une approche à base d'agents mobiles et stationnaires, qui coopèrent pour calculer l'ensemble des itemsets localement fréquents sur chaque site du réseau (ou sur chaque fragment de l'entrepôt de données global), l'ensemble des itemsets globalement fréquents sur la totalité des données l'entrepôt global et enfin, l'ensemble de règles d'association.

Afin de comprendre la notion d'agents mobiles et stationnaires, nous présentons dans cette annexe quelques terminologies liées aux systèmes multi-agents, aux agents mobiles et aux aglets (la plateforme la plus populaire des agents mobiles).

II Définitions

II. 1 Agents

Dans la littérature scientifique, on trouve une multitude de définitions pour la notion d'agent. Les plus réputées est celles données par Ferber [Fer 1995] et Jennings et Wooldridge récemment [Jen 2000] :

D'après Ferber [Fer 1995], « un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents ».

D'après [Jen 2000], « un agent est un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu ».

- *Situé* dans un environnement particulier et capable d'agir sur lui ; il reçoit des entrées liées aux états de cet environnement par des capteurs et agit sur cet environnement par des émetteurs.

- *Autonome* ; capable d'agir sans l'intervention directe d'un tiers (humain ou agent) et a un certain contrôle sur ses propres actions et sur son état interne ;
- *Flexible* ; capable d'adopter un comportement flexible pour résoudre des problèmes selon les objectifs pour lesquels il a été conçu. Il est *réactif* (capable de percevoir son environnement et répondre dans le temps requis aux changements qui celui-ci peut entraîner sur les agents), *proactif* (capable de prendre des initiatives pour atteindre ses propres buts et objectifs dont il a capacité de se fixer) et *social* (capable d'interagir avec les autres agents afin d'accomplir ses tâches ou aider ces agents à accomplir les leurs).

II. 2 Agents mobiles et stationnaires

Selon la mobilité, on distingue deux types d'agents [Sek 2007] [Zga 2007] :

Agents mobiles

Un agent mobile est composé de code (le programme ou l'algorithme réalisant ses tâches assignées), de données (les données utilisées pour réaliser ses tâches) et d'un état d'exécution (valeurs des registres, pile d'exécution, etc.). Il dispose de la capacité de migrer (avec son code, données et état d'exécution) d'un nœud à un autre dans un réseau pour s'exécuter et donc d'accomplir un certain nombre de tâches en utilisant les ressources des nœuds visités. Après exécution, l'agent mobile retourne éventuellement vers son site d'origine afin de lui fournir les résultats de son exécution (voir la figure 7. 1)

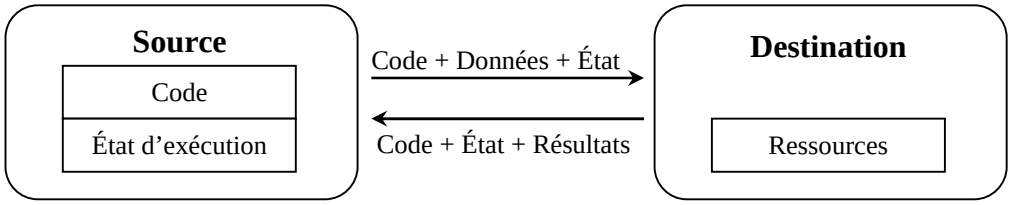


Figure 1 Mécanisme de migration d'agents mobiles (*migration forte*)

Il existe généralement deux modes de migration d'agents mobiles [Zga 2007] :

- *La migration forte*, où la totalité de l'agent (c'est-à-dire code, données et état d'exécution) migre vers le site distant. Dans ce cas, à l'arrivée au site distant, l'agent mobile redémarre son exécution (restauration) au point de contrôle précédent, en conservant son état d'exécution.
- *La migration faible*, où seul le code et les données migrent. Une fois arrivé sur le site distant, l'agent mobile réexécute son code.

Agents stationnaires

Les agents stationnaires ou statiques sont des agents mobiles qui n'exécutent en général qu'une seule migration. Cette migration s'effectue depuis son site initiateur (où l'agent stationnaire est crée et initialisé) vers un nœud bien défini au lancement. À l'arrivée sur le nœud spécifié, l'agent stationnaire ne migre plus mais exécute une tâche prédéfinie. Ces agents sont généralement utilisés pour servir des itinéraires pour les agents mobiles de la plateforme d'agents mobiles utilisée.

II. 3 Système multi agents

Un Système Multi-Agent (SMA) est un système logiciel distribué composé d'un ensemble d'agents qui interagissent le plus souvent, selon des modes de coopération, de concurrence ou de coexistence.

Un SMA est généralement caractérisé par [Cha 1999] :

- Chaque agent a des informations ou des capacités de résolution de problèmes limitées ;
- Chaque agent a un point de vue partiel de son environnement ;
- Il n'y aucun contrôle global du système multi-agents ;
- Les données sont décentralisées ;
- Le calcul est asynchrone.

III La plateforme Aglets

Un aglet est la contraction d'**Ag**ent et App**let**. IBM a confié à une équipe japonaise le soin de développer une technologie pour les agents mobiles. Aglets est le résultat des recherches de cette équipe. Il s'agit d'une API java spécifiant un ensemble de méthodes répondant aux principes fondamentaux des agents mobiles.

III. 1 Définitions

Les principaux éléments de l'architecture aglet sont [Sek 2007] :

- **Aglets** (**Ag**ents App**lets**) : Les aglet sont des objets java mobiles qui peuvent se déplacer d'une machine hôte à une autre. Ainsi, un aglet qui s'exécute sur un hôte peut stopper son exécution, se déporter vers un hôte distant et continuer cette exécution dans son nouvel environnement.

- **Proxy** : Les méthodes d'un aglet ne sont jamais invoquées directement. Elles sont invoquées au travers un objet AgletProxy (voir la figure 7. 2). Cet objet est obtenu en retour lors de la création d'un aglet ou à l'aide de l'identificateur de l'aglet.
- **Contexte** : Le contexte représente l'environnement d'exécution de l'aglet. Il offre des moyens pour mettre à jour et contrôler l'aglet dans un environnement uniforme d'exécution. Dans le même contexte, plusieurs aglets peuvent être crées.
- **Hôte** : Un hôte est une machine capable d'héberger plusieurs contextes. L'hôte est généralement un nœud dans un réseau.
- **Identificateur** : À chaque aglet est attribué à nom globalement unique qui permettra de le localiser dans n'import quel contexte.
- **Message** : Un message est un objet échangé entre les aglets via un proxy.

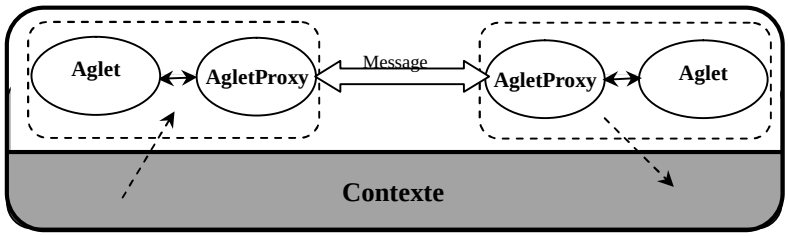


Figure 2 Relation entre Aglet, AgletProxy et contexte

III. 2 Cycle de vie d'un aglet

Depuis sa création jusqu'à sa terminaison, un aglet peut passer par plusieurs étapes (cycle de vie d'un aglet). Les principales opérations affectant la vie d'un aglet sont (voir la figure 7. 3) :

- **Création** : La création d'un aglet se fait dans un contexte. Un nom globalement unique est attribué à l'aglet nouvellement crée. Ce dernier permettra de le localiser dans n'import quel contexte et de communiquer avec lui. Une fois crée, l'aglet peut être actif, c'est-à-dire que les initialisations nécessaires à l'exécution de son code sont effectuées et son exécution est déclenchée.

  La création d'un aglet se fait par l'appel de la méthode « *createAglet* (URL codeBase, String code, Object init) ». Cette méthode retourne le proxy associé à l'aglet nouvellement crée.
- **Clonage** : Le clonage d'un aglet produit une copie presque identique de l'aglet original dans le même contexte. Un nom différent est assigné à l'aglet crée par l'opération de clonage et son exécution sera relancée.

La méthode « *clone* () » permet de cloner un aglet. Cette méthode retourne le proxy associé à l'aglet clone (c'est-à-dire l'aglet crée par l'opération de clonage).

- **Déportation (ou Dispatching)** : La déportation d'un aglet provoque son déplacement d'un contexte vers un autre. Précisément, dispatcher un aglet entraînera sa suppression de son contexte courant et son insertion dans son contexte de destination, où il continuera son exécution.

  La méthode « *dispatch* (URL destination) » permet de déporter un aglet vers un autre contexte. Si appliquée au proxy, cette méthode retourne le nouveau proxy associé à l'aglet.

- **Récupération (Rétractation)** : L'aglet déporté est récupéré dans son contexte d'origine. Autrement dit, le retrait d'un aglet va le supprimer de son contexte courant et l'insérer dans le contexte où l'opération a été invoquée.

  Le retrait d'un aglet s'effectue par l'appel de la méthode « *retractAglet* (URL contextAddress, AgletID aid) » qui consiste à retirer un aglet de son contexte actuel et de le ramener vers son contexte d'origine. Si appliquée au proxy, cette méthode retourne le nouveau proxy associé à l'aglet.

- **Activation et Désactivation** : La désactivation d'un aglet est une interruption temporaire de son exécution. Elle consiste à le stocker momentanément dans un espace de stockage secondaire afin de pouvoir le réactiver ultérieurement.

  La désactivation d'un aglet s'effectue par la méthode « *deactivate* (long duration) » et son réactivation s'effectue par la méthode « *activate* () ».

- **Libération ou destruction** : Cette dernière opération marque la fin du cycle de vie de l'aglet en le supprimant de son contexte d'exécution courant.

  La méthode « *dispose* ()» permet de supprimer un aglet dans ce contexte courant.

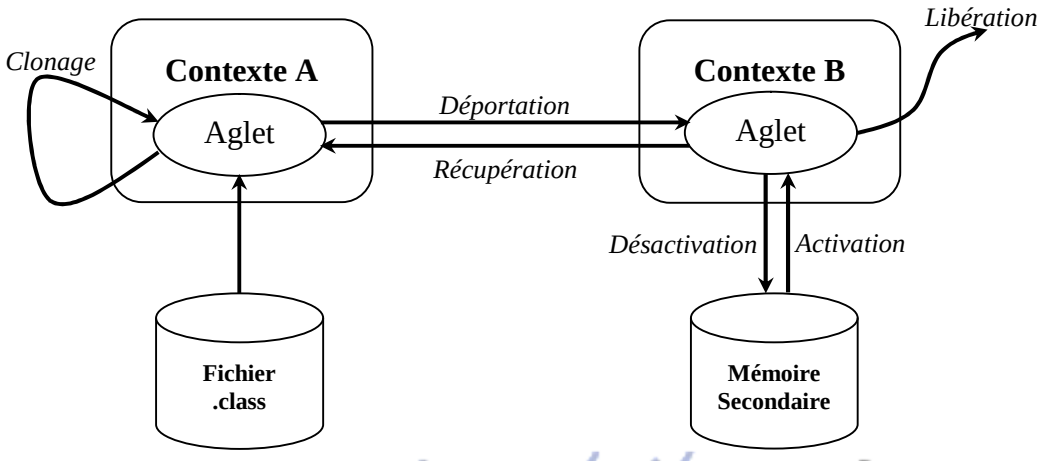


Figure 3 Cycle de vie d'un aglet

III. 3 Communication entre aglets

La communication est basée sur l'échange d'objets de la classe « *Message* ». Ce modèle de communication est indépendant de la localisation de l'aglet et peut être asynchrone ou synchrone.

Un aglet désirant envoyer un message doit obligatoirement passer par le proxy du destinataire. En fait, le proxy reste l'intermédiaire obligatoire pour tout échange.

Chaque aglet possède un gestionnaire de messagerie « *MessageManager* » qui lui permet de les traiter un à un et dans l'ordre de leurs arrivées respectives. Toutefois, cet ordre peut être changé par l'aglet en modifiant les priorités des messages dans la file d'attente. Ceci est possible grâce à la méthode « *setPriority* (String kind, int priority) ».

Chaque message est caractérisé par la catégorie « Kind » à laquelle il appartient et au contenu « Arg » de n'importe quel type. La création d'une instance message nécessite l'affectation d'une valeur à son paramètre « Kind ».

La méthode « *Message* (String Kind, Object Arg) » permet de créer un nouveau message de type « kind » avec un argument de type « Object ».

Messages synchrones

La méthode « *sendMessage* (Message msg) » permet l'envoie de messages synchrones, elle est donc bloquante car elle attend la réponse du récepteur. Le destinataire du message doit implémenter un gestionnaire des messages reçus. Cette fonctionnalité est rendue possible par la surcharge de la méthode « *handelMessage*(Message msg) ». La méthode « *sendReply* () » permet de retourner une réponse à l'émetteur.

Messages asynchrones

La messagerie asynchrone est implémentée grâce à la notion de « *futur objet* ». L'envoi d'un message asynchrone retourne un lien vers la réponse même si cette dernière n'existe pas encore. Ce lien permet de tester si la réponse est arrivée ou pas. Cette technique permet à l'aglet d'envoyer un message sans être obligé d'interrompre son exécution en attente de la réponse.

Exemple de méthodes qui permettre de tester l'arrivée d'une réponse sont : La méthode « *isAvailable* () », permet de tester si la réponse est arrivée et la méthode « *waitForReply* () » permet d'attendre pendant une période indiquée afin de tester si la réponse est arrivée.

Résumé

L'Extraction Parallèle et Distribuée de Connaissances (*EPDC*) est un domaine de recherche très récent qui s'intéresse au développement des solutions parallèles et distribuées vis-à-vis de l'extraction de connaissances à partir des sources de données distribuées et hétérogènes. Dans ce travail de magistère, nous nous sommes intéressés particulièrement aux ressources de stockage de type entrepôts de données et aux connaissances exprimables sous forme de règles d'association, en proposant une approche parallèle et distribuée d'extraction de règles d'association à partir d'un entrepôt de données réparti. Due aux intérêts apportés par le concept d'agent mobile aux résolutions des problèmes distribués, l'approche proposée dans ce rapport est s'appuie sur un ensemble d'agents mobiles et stationnaires qui coopèrent pour calculer les connaissances locales à chaque site du réseau (itemsets localement fréquents) et les connaissances globales sur la totalité des données de l'entrepôt (itemsets globalement fréquents et règles d'association). En outre, la génération des itemsets fréquents et leurs mises à jour sur chaque site du réseau s'effectuent d'une façon parallèle en exploitant l'algorithme Partition et l'algorithme Incremental Mining, pour lesquels nous avons proposés des améliorations avant de les adaptés dans l'approche proposée.

Mot Clés : *Entrepôt de données, Fouille de données, Techniques d'extraction de connaissance, Extraction Parallèle et Distribuée de connaissance, Approche à base d'agents mobiles d'extraction parallèle et distribuée de règles d'association*

Abstract

Parallel and Distributed Knowledge Discovery (*PDKD*) is a very recent field of research which is interested in the development of parallel and distributed solutions towards the extraction of knowledge from the distributed and heterogeneous data sources. In this work, we were interested particularly in the resources of storage of the type Data Warehouses and in knowledge expressible in the form of association rules, by proposing an approach of parallel and distributed mining of association rules from a distributed Data Warehouse. Because of the interests brought by the mobile agent concept to the resolutions of the distributed problems, the approach proposed in this report is based on a set of mobile and stationary agents which cooperate to calculate local knowledge in each site of the network (itemsets locally frequent) and global knowledge on the totality of the data of the warehouse (itemsets globally frequent and association rules). Moreover, the generation of the frequent itemsets and their updates on each site of the network are carried out in a parallel way by exploiting the Partition algorithm and the Incremental Mining algorithm, for which we proposed improvements before adapted in the approach proposed.

Keywords : *Data Warehouse, Data Mining, Knowledge Discovery techniques, Parallel and Distributed Knowledge Discovery, Approach based mobiles agents of parallel and distributed mining of association rules.*