N°d'ordre : 01/2013-D/Inf

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et Technologies Houari Boumediene

Faculté d'Electronique et d'Informatique
Département d'Informatique

THESE

Pour l'obtention du Grade de

Docteur

en

Informatique

Présentée par

HAMID NECIR

Intégration d'Agents Intelligents dans les Approches Data Mining pour la Sélection des Index et des Vues Matérialisées dans les Entrepôts de Données Relationnels

Soutenue publiquement le 27 mars 2013

Devant le jury composé de :

AISSANI MOKHTARI	Aicha	Professeur	à l'U.S.T.H.B	Présidente
DRIAS	Habiba	Professeur	à l'U.S.T.H.B	Directrice de thèse
HADJ ALI	Allel	Professeur	à l'Université de Renne	Examinateur
ZEGOUR	Djamel	Professeur	à l'E.S.I	Examinateur
BOUKHALFA	Kamel	MCA	à l'U.S.T.H.B	Examinateur
HIDOUCI	Walid-Khaled	MCA	à l'E.S.I	Examinateur
BELLATRECHE	Ladjel	Professeur	E.N.S.M.A, Poitiers	Membre invité

Dédicace

A ma très chère mère ouerdya

Tous les mots du monde ne sauraient exprimer l'immense amour que je te porte, ni la profonde gratitude que je te témoigne pour tous les efforts et les sacrifices que tu n'as jamais cessé de consentir.

Je te rends hommage par ce modeste travail en guise de ma reconnaissance éternelle et de mon infini amour.

Qu'ALLAH tout-puissant te garde et te procure santé, bonheur et longue vie pour que tu demeures à jamais ce flambeau illuminant notre chemin.

Hamid

Remerciements

En tout premier lieu, je tiens à exprimer ma reconnaissance à Madame Drias Habiba (Professeur à l'Université des Science et de la Technologie Houari Boumediene) pour avoir encadré cette thèse. Ses qualités humaines et scientifiques ainsi que ces conseils judicieux m'ont permis de mener à bien et avec grand plaisir ces travaux et je l'en remercie très chaleureusement.

Je remercie l'ensemble des membres du jury : Madame Aissani-Mokhtari Aicha (Professeur à l'Université des Science et de la Technologie Houari Boumediene) pour avoir accepté de le présider.

Messieurs Hadjali Allel (Professeur à l'université de Renne, France), Zegour Djamel (Professeur à l'Ecole Supérieure d'Informatique de oued smar), Hidouci Walid-Khaled (Maitre de Conference/A à à l'Ecole Supérieure d'Informatique de oued smar) et Boukhalfa Kamel (Maitre de Conference/A à l'Université des Science et de la Technologie Houari Boumediene), qui m'ont fait l'honneur d'accepter de lire et de juger cette thèse.

Je tiens aussi à exprimer mes vives remerciements à Monsieur Bellatreche Ladjel (Professeur à l'ENSMA de Poitiers), avec qui j'ai eu le plaisir et l'honneur de collaborer sur un certain nombre de travaux et enfin pour sa présence en tant que membre invité de ce jury. Qu'il veuille trouver ici l'expression de mes remerciements les plus sincères

Je remercie infiniment ma mère de m'avoir toujours soutenue et pour m'avoir toujours encouragé à aller le plus loin possible. Mes sincères remerciements vont aussi à mes frères et à mes soeurs qui ont toujours été là pour partager mes soucis et mes joies.

Je ne remercierais jamais assez mon épouse Amina pour sa patience, son soutien, ses encouragements quotidiens, son écoute et son enthousiasme... Je lui dois une très profonde gratitude.

Je voudrais également avoir ici une pensée des plus affectueuses pour mon père qu'il repose en paix dans son vaste paradis.

Merci à tous ceux et celles qui ont contribué de prés ou de loin à la réussite de ce travail et dont je ne citerais pas les noms de peur d'en oublier certains.

Resumé

Actuellement, les entrepôts de données représentent la meilleure solution pour la mise en œuvre de systèmes permettant de stocker de gros volumes de données, organisée de façon à en faciliter l'analyse et la prise de décision. Cependant, étant donné leur grande volumétrie et la complexité de leurs requêtes d'interrogation, caractérisée par le nombre élevé des opérations de jointures et d'agrégations. Ces systèmes ne peuvent être gérés convenablement sans l'utilisation de structures d'optimisation permettant l'amélioration de leur performance.

Dans cet objectif, plusieurs techniques ont été proposées. Ces techniques peuvent être classées en deux catégories : (1) les techniques d'optimisation non redondantes comme la fragmentation horizontale et le traitement parallèle et (2) les techniques d'optimisation redondantes comme les index et les vues matérialisées. Néanmoins, l'exploitation de ces techniques d'optimisation redondantes nécessite un espace de stockage supplémentaire et entraîne des coûts de maintenance lors de l'actualisation des données.

Dans cette thèse, nous nous intéressons aux structures redondantes, à savoir les index et les vues matérialisées étant donné que ces structures sont considérées comme très performantes pour optimiser l'administration des entrepôts données.

Nous développerons de nouvelles approches qui comblent les lacunes des méthodes existantes. Nos stratégies utilisent des techniques data mining distribuées à base d'agents intelligents. Nous nous intéresserons aussi à la distribution de l'espace entre ces deux structures dans le cas statique.

Dans le cadre de la sélection d'index, nous étudions et montrons les insuffisances des approches de sélection à base data mining. Par la suite, nous proposons, pour la sélection d'index de jointure binaires, une approche basée sur la recherche distribuée des itemsets fréquent fermé utilisant un système multi agents coopératif et une contrainte d'élagage convertible anti monotone renforcée,

Dans le cadre de la sélection des vues matérialisées, nous préconisons d'adopter une approche de clustering distribuée à base d'agents coopératifs. Une fois les candidats générés, nous exploitons, pour les deux techniques d'optimisation, des modèles de coûts afin d'évaluer et de choisir, sous une contrainte d'espace de stockage, une configuration pertinente d'index de jointure binaires ou de vues matérialisées. Enfin, nous abordons le problème de partage de l'espace de stockage entre les index et les vues matérialisées.

Mots clés : Agents, Data mining, Entrepôts de données, Index, Itemsets fréquents, Vues matérialisées.

Abstract

The amount of information in a data warehouse tends to be extremely large and queries may involve several complex join and aggregates operations at the same time. To improve performance of these queries, database administrators often use optimization structures. We can classify them into two main categories : (1) non redundant structures and (2) redundant structures. Redundant structures, like materialized views and bitmap join indexes are more efficient to enhance query execution in data warehousing

Their main drawbacks are the extra storage requirement and the maintenance overhead. In this thesis, we are interested in (1) the selection of bitmap join index, (2) the selection of materialized views and (3) the problem of space distribution among views and indexes in the static cases.

To deal with the selection of bitmap join indexes or materialized views, we propose two new approaches with two main phases. The first involves pruning the search space to reduce the number of candidates. In this order, we use a distributed data mining approach using multi agents system that can significantly reduces the complexity of the selection process. The second phase uses also a multi agent's architecture to select the final set that minimizes the query processing cost and satisfy the storage constraint.

For selecting bitmap join indexes, we use closed frequent itemsets that representing candidate attributes for the index selection process. The main particularity of our pruning approach, compared to the existing ones, is that it uses multi agents and a convertible anti-monotone constraint.

To deal with the selection of materialized views, we use a distributed clustering approach using multi agents system that can significantly reduces the complexity of the selection process. We use also a multi agent's architecture to capture the relationships between views candidates to select the final set of materialized views.

Finally, we propose a new approach to automatically distribute the storage space among views and indexes. This approach uses an adapted *FIPA-Contract-Net* protocol to correctly distribute the storage space between participants agents (one for views and another for indexes).

Keywords : Data mining, Data warehouse, Frequent itemset, Index, Multi agents, Materialized view.

Table des figures

Liste des tableaux

Liste des algorithmes

Table des matières

Introduction générale

0.1 Contexte d'application

Les entrepôts de données et les bases de données de grande taille sont souvent accédés par des requêtes complexes et coûteuses en termes de temps de calcul par le fait qu'elles nécessitent des opérations de jointures et d'agrégations.

Les entrepôts de données sont souvent représentés par un schéma en étoile constitué par une table des faits et de tables de dimension. Les requêtes typiques définies sur ce schéma sont appelées les requêtes de jointure en étoile (star join queries) qui ont les caractéristiques suivantes : (1) elles possèdent des jointures multiples entre la table des faits ayant une taille importante et les tables de dimension, (2) elles n'ont aucune jointure entre les tables de dimension (toute opération de jointure passe par la table centrale qui est la table des faits) et (3) chaque table de dimension impliquée dans une opération de jointure possède plusieurs prédicats de sélection sur ses attributs descriptifs.

Dans le contexte des entrepôts de données relationnels, plusieurs structures ont été proposées pour optimiser les performances et réduire le temps de réponse des requêtes d'interrogation. Ces dernières peuvent être classées en deux catégories : (1) les structures d'optimisation non redondantes et (2) les structures d'optimisation redondantes. Dans la première catégorie, nous pouvons citer la fragmentation et le traitement parallèle. Dans la seconde catégorie, nous pouvons citer les index et les vues matérialisées. Comme leur nom l'indique, ces structures nécessitent un espace de stockage et un coût de maintenance.

Dans le cadre de ces travaux nous nous intéressons aux structures redondantes, à savoir les index et les vues matérialisées. Etant donné que ces structures sont considérées comme très performantes pour optimiser les performances dans le contexte des entrepôts données relationnels [181]. Nous développerons de nouvelles approches qui comblent les lacunes des méthodes existantes. Nos stratégies utilisent des techniques data mining distribuées à base d'agents intelligents.

L'indexation est l'une des techniques d'optimisation redondante qui accélère les requêtes *OLAP*. Deux types d'index sont disponibles : les mono-index (B-arbre, index binaires, projection etc.) et les multi-index (index de jointure). L'index de jointure binaire est multi tables. Ceci le rend très adapté pour optimiser les requêtes *OLAP* dans le contexte d'un entrepôt représenté avec un schéma en étoile. Il permet de pré-calculer les jointures en stockant sous format binaire toute combinaison de clés étrangères de la table des faits. Une autre caractéristique offerte par les index de jointure binaires est la compression [126], d'où une réduction de leur espace de stockage et la possibilité de les stocker en mémoire

centrale. Notons que les index de jointure binaires sont particuliérement recommandés pour des attributs ayant une faible cardinalité comme genre.

Contrairement à une vue classique qui est calculée à chaque consultation à partir de la base de données, une vue matérialisée est définie comme une table contenant les résultats d'une requête. Les vues améliorent l'exécution des requêtes en pré calculant les opérations les plus coûteuses comme la jointure et l'agrégation et en stockant leurs résultats dans une table [108]. En conséquence, certaines requêtes nécessitent seulement l'accès aux vues matérialisées et sont ainsi exécutées plus rapidement [20]. Plusieurs travaux et expériences ont montré que l'utilisation judicieuse des vues matérialisées permet d'améliorer le temps de traitement des requêtes complexes avec des ordres de grandeurs significatifs [13], [53], [116], [181].

Actuellement, la plupart des *SGBDs* commerciaux supportent l'utilisation des index de jointure binaires et des vues matérialisées. Cependant, vu le nombre important d'attributs candidats participant à la construction des index et l'impossibilité de matérialiser toutes les vues, une des tâches les plus importantes de l'administrateur est de sélectionner un ensemble d'index et de vues permettant d'améliorer au mieux les performances [2].

Cette sélection qui se fait généralement de façon séparée, pose un problème de gestion de ressources dans le cas où on a une contrainte sur la capacité d'espace qu'il faut correctement distribuer entre les vues et les index afin de garantir une meilleure performance des requêtes.

0.2 Objectifs et contributions

Ce travail de thèse s'intéresse à l'optimisation des performances des entrepôts de données sous l'angle de la sélection des structures d'optimisation redondantes. Ces structures sont très efficaces pour améliorer les performances et réduire le temps de réponse des requêtes d'interrogation *OLAP*.

Notre objectif principal consiste à proposer des méthodes à base des techniques data mining pour réduire la complexité du problème de sélection dans le contexte des entrepôts de données modélisés par un schéma en étoile. Plus précisément, le travail contenu dans cette thèse apporte des contributions autour de trois axes principaux :

- la proposition d'une approche data mining pour la sélection d'index de jointure binaires, basée sur la recherche distribuée des itemsets fréquents fermés utilisant un système multi agents coopératif,
- la proposition d'une approche data mining pour la sélection de vues matérialisées basée sur la classification hiérarchique distribuée et utilisant un système multi agents coopératif,
- la proposition d'une approche pour le partage de l'espace de stockage entre les index de jointure binaires et les vues matérialisées.

Plusieurs travaux de recherche ont montré l'utilité des index et des vues matérialisées et proposent différentes approches pour la sélection de ces structures dans le contexte des bases de données ou des entrepôts de données. Cependant, peu de travaux se sont intéressés à l'utilisation des techniques data mining dans le cadre de la sélection des index et des vues matérialisées. Cette option reste très intéressante étant donné que contrairement

aux travaux précédents, elle prend en compte les connaissances qui peuvent être extraites directement de l'entrepôt et de la charge de requêtes (métadonnées, statistiques, usage des attributs etc.) afin de réduire la complexité du problème de sélection et de cibler les vues et les index candidats les plus pertinents [7].

Nous étendons ces travaux en montrant leurs limites que ce soit dans le cadre de la sélection d'index ou de vues matérialisées. L'idée de base des stratégies que nous proposons est d'adapter les algorithmes data mining au problème de sélection d'index et de vues matérialisées.

Dans le contexte de sélection d'index, nous montrons que la principale limitation des précédents travaux est qu'ils considèrent seulement une démarche centralisée avec seulement la fréquence d'accès des requêtes comme métrique d'élagage.

Ce choix peut être discutable car la fréquence ne garantit pas à elle seule la qualité de l'index sélectionné puisque le coût d'une opération de jointure dépend fortement de la taille des tables jointes [93]. Beaucoup de recherches ont montré la faiblesse des algorithmes qui utilisent uniquement le paramètre de fréquence comme métriques d'élagage [115], [87].

Partant de ces constats, nous proposons une stratégie de sélection d'index qui prend en considération des paramètres autres que la fréquence d'accès, comme la taille des tables (la table des faits et les tables de dimension) concernées par l'indexation.

En nous basant sur les travaux de [173], nous améliorons notre démarche de sélection distribuée utilisant un ensemble d'agents intelligent afin de réduire de façon significative la complexité de ce problème. Nos agents utilisent un critère d'élagage sous forme d'un facteur pénalisant relatif à la cardinalité et au nombre de pages des différentes tables de dimension par rapport à la table des faits et qui respecte en même temps le critère de monotonie et d'anti monotonie.

Pour construire un ensemble des vues candidates, nous considérons le schéma en étoile comme une vue conceptuelle sur laquelle des vues matérialisées peuvent être introduites pour accélérer les traitements. Nous préconisons dans cette étape une stratégie de sélection utilisant la classification non supervisée à base d'agents coopératifs. Cette démarche est motivée pour deux raisons : (1) la classification non supervisée permet de regrouper les parties communes qui existent dans plusieurs requêtes ayant une syntaxe tres similaire et qui sont susceptibles d'être optimisées à partir d'une même vue matérialisée. (2) une approche de classification non supervisée classique demande un temps de calcul extrêmement élevé [149], ce qui la rend inadaptée dans le contexte des entrepôts de données caractérisés par le grand volume des données et de la complexité des requêtes d'interrogation.

Une fois les candidats générés, nous exploitons, pour les deux stratégies de sélection, des modèles de coûts afin d'évaluer et de choisir, sous une contrainte d'espace de stockage, une configuration pertinente d'index de jointure binaires ou de vues matérialisées.

Par ailleurs, nous abordons le problème de partage de l'espace de stockage entre les index et les vues matérialisées en assimilant notre distribution à un problème de partage d'une ressource s'inspirant du protocole *FIPA-Contract-Net* [82].

0.3 Organisation de la thèse

Après cette introduction qui nous permet de situer notre travail. Cette thèse va s'organiser comme suit :

La partie I qui s'intitule état de l'art contient les chapitres suivants :

- Le Chapitre 1 réalise un tour d'horizon des techniques d'optimisation des entrepôts de données, en se focalisant plus particulièrement sur les principaux travaux proposés pour résoudre le problème de sélection d'index et de vues matérialisées.
- Le Chapitre 2 présente le data mining et particulièrement l'approche de clustering et les règles d'association basée sur le treillis de concepts.
- Le chapitre 3 nous permet de donner un aperçu sur le système multi agents et son intérêt dans la mise en place d'une démarche data mining distribuée.

La partie II qui s'intitule contribution contient les chapitres suivants :

- Le Chapitre 4 définit le problème de sélection des index de jointure binaires et le replace dans la thématique plus générale du data mining. Il présente les motivations et les étapes de notre stratégie de sélection d'index basée sur la recherche des itemsets fréquents fermés utilisant un ensemble d'agents intelligent. Nous présentons aussi un ensemble de résultats expérimentaux confirmant la justesse de nos choix.
- Le Chapitre 5 développe en détails notre stratégie de sélection de vues matérialisées basée sur la classification non supervisée à base d'agents. Nous abordons également ici les motivations qui nous ont amenées à choisir la classification distribuée comme heuristique permettant de réduire la complexité du problème de sélection. Nous présentons à la fin nos résultats expérimentaux.
- Le Chapitre 6 présente les motivations et les étapes de notre stratégie de partage de l'espace de stockage entre les index et les vues matérialisées en adaptant le protocole *FIPA-Contract-Net*. Nous présentons aussi un ensemble de résultats expérimentaux confirmant la justesse de nos choix.

Enfin, on conclut cette thèse en dressant le bilan de nos contributions et en indiquant les perspectives de recherche qui en découlent.

Première partie

Etat de l'art

Chapitre 1

Entrepôt de données et structures d'optimisation

1.1 Introduction

Actuellement, les entrepôts de données représentent la meilleure solution pour la mise en œuvre de systèmes permettant de stocker de gros volumes de données, organisée de façon à en faciliter l'analyse et la prise de décision.

Cependant, étant donné la complexité des requêtes d'interrogation, caractérisées par le nombre élevé des opérations de sélections, de jointures et d'agrégations. Ces systèmes ne peuvent être gérés convenablement sans l'utilisation de structures d'optimisation permettant l'amélioration des performances.

Dans ce chapitre, nous présentons un état de l'art portant sur les entrepôts de données et les différents types de modélisation multidimensionnelle. Nous décrivons par la suite les principales techniques d'optimisations permettant d'améliorer le temps d'exécution des requêtes décisionnelles dans le contexte relationnel, leurs problèmes de sélection ainsi que les principaux travaux s'intéressant à ces techniques.

1.2 Architecture d'un entrepôt de données

Un entrepôt de données peut être vu comme une base de données dont l'information, généralement volumineuse et organisée de façon à en faciliter l'analyse et la prise de décision.

Bill Inmon dans son ouvrage de référence [121] définit l'entrepôt de données comme une collection de données orientées sujet, intégrées, non volatiles et historisées, organisées pour le support d'un processus d'aide à la décision.

Nous pouvons déduire de cette définition qu'un entrepôt de données possède les caractéristiques suivantes :

- *Données orientées sujet :* L'entrepôt de données est organisé autour des sujets majeurs de l'entreprise (tels que les clients, ventes) au lieu d'être organisé selon les domaines d'application (tels que la facturation des clients, gestion de stock). L'intérêt

FIGURE 1.1: Architecture d'un entrepôt de données

de cette organisation est de pouvoir disposer de l'ensemble des informations utiles sur un sujet et permettre ainsi de développer le système décisionnel.

- *Données intégrées.* Comme les données proviennent de sources différentes, elles sont de ce fait souvent incohérentes, puisqu'elles arborent des formats différents. Elles doivent être mises en cohérence afin de présenter une vue unifiée des données. Cette étape d'intégration est indispensable car les informations communes à plusieurs sujets ne doivent pas être dupliquées.
- *Données historisées.* Puisque le data warehouse représente les données d'une entreprise sur une longue période et que certaines ne sont plus valables après un certain intervalle de temps. Il faut gérer les différentes valeurs que celles-ci peuvent prendre. Cela permet d'effectuer un suivi et une analyse de ces dernières.
- *Données non volatiles.* Les données présentes dans les entrepôts de données ne peuvent être modifiées mais uniquement rafraichies à partir du système, les nouvelles informations viennent en supplément.
- *Aide à la décision.* Un entrepôt de données est destiné à l'aide à la décision. Son objectif principal est d'être une plate forme commune pour supporter les opérations et requêtes complexes de type *OLAP* (On Line Analytical Processing). Ces requêtes accèdent généralement à un grand volume de données et effectuent un grand nombre de jointure et d'agrégation. Ainsi la création d'un entrepôt passe par un ensemble d'étapes commençant par l'extraction des données dans les bases de production, leurs nettoyages, leurs normalisations, puis leurs intégrations.

Des métadonnées sont aussi créés afin de décrire les informations relatives au système d'information qui régit l'entrepôt. Ces informations permettent de lever toute ambiguïté sur chaque valeur en la définissant par une description précise et claire. Cette description porte entre autres sur les tables et leurs champs, les utilisateurs, les autorisations, .etc.

1.3 Modélisation d'un entrepôt de données

Les différences qui distinguent un entrepôt de données d'un système opérationnel sont à la fois d'ordre conceptuel, car l'objectif est différent et d'ordre technique, car la structure et les processus sont différents [131].

Ainsi, au moment où les applications *OLTP* (On-Line Transaction Processing) sont la cible essentielle des systèmes de bases de données transactionnelles basées sur le modèle

Entité-Relation décrivant des relations entre données élémentaires en cherchant l'élimination des redondances. Ce modèle est caractérisé par des requêtes d'interrogation peu complexes et des mises à jour ponctuelles des données.

Les applications *OLAP* (On-Line Analytical Processing) sont nés de l'accroissement des volumes de données et de l'évolution des besoins des entreprises qui souhaitent avoir un point de vue global pour manipuler des sources de données hétérogènes, sur lesquelles des traitements analytiques sont associés aux traitements transactionnels. Les techniques *OLAP* permettent, à des fins d'analyse, de collecter, stocker, traiter et présenter de façon appropriée les données multi-dimensionnelles [3], selon différentes dimensions et selon différents degrés de granularité.

Caractéristiques	OLTP	OLAP
Modélisation	entité-relation	Étoile, flocon, constellation
Fonction	Gestion courantes	Aide à la décision
Données	Actuelles, Brutes	Historiques, agrégées, intégrées, consolidées
Mise à jour	Immédiate et fréquentes	Différées et peu fréquentes
Opérations	Lecture et écriture	Lecture et rafraîchissement
Taille	En giga octets	En Téra octets
Utilisateur	Employé	Décideur et analyse
Perception	Bidimensionnelles	Multi-dimensionnelles
Requête	Simples	Complexes

TABLE 1.1: Les différentes caractéristiques entre le modèle OLTP et OLAP

Toutes ces raisons ont poussé les praticiens [131], [121] et les chercheurs [3], [200] à adopter ce modèle pour organiser les données d'un entrepôt. Le concept de base est appelé cube de données [98].

Un cube peut être vu comme un ensemble de cellules, chacune d'elle représentant une association entre un élément de chaque dimension (appelé membre), d'une part et un contenu de cellule (appelé mesure), d'autre part.

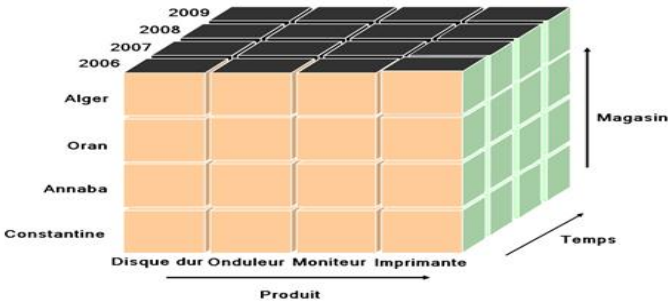

FIGURE 1.2: Cube de donné modélisant des ventes sous forme Multi-dimensionnelle

Pour construire ce cube, deux approches existent : l'approche *MOLAP* (Multidimensional *OLAP*) et l'approche *ROLAP* (Relational *OLAP*) qui utilise un *SGBD* relationnel pour stocker le cube de données [20]. Le tableau ci-dessous montre quelques solutions *OLAP* présentes sur le marché.

Nom	Editeur	Technologie
DB2 UDB Server	IBM	ROLAP
Oracle11g	Oracle	ROLAP
SQL Server 2010	Microsoft	ROLAP
DSS	Microstrategy	ROLAP
TeraData	Teradata Corporation	ROLAP massivement parallèle
Informix Metacube	Informix	MOLAP
Essbase	Arbor Software/Hyperion	MOLAP
SAS OLAP Server	SAS	MOLAP
Metacube	Informix	ROLAP
SQL Server	Microsoft	HOLAP
MDDB	SAS Institute	MOLAP/ROLAP
Oracle Express-server	Oracle	MOLAP/ROLAP
DB2 OLAP Server	IBM	MOLAP/ROLAP
Crystal	Seagate Software	Serveur d'application OLAP unique pour tous les déploiements
PowerPlay	Cognos	Serveur d'application OLAP unique pour tous les déploiements

TABLE 1.2: Quelques solutions *OLAP* présentes sur le marché

1.3.1 L'approche *MOLAP*

Les systèmes *MOLAP* stockent les données dans un *SGBD* multidimensionnel sous la forme d'une matrice multidimensionnelle avec un mode de stockage optimisé. Chaque dimension de ce tableau est associée à une dimension du cube. Seules les valeurs correspondantes aux données de chaque cellule sont stockées.

Ces systèmes demandent un pré-calcul de toutes les agrégations possibles. Ainsi, toute valeur d'indicateur associée à l'axe temps sera pré-calculée au chargement pour toutes ses valeurs hebdomadaires, mensuelles etc.

En conséquence, ils sont plus performants et efficaces que les systèmes traditionnels pour le traitement des requêtes mais plus difficiles à mettre à jour. Plus le volume de données à gérer est important, plus les principes d'agrégations implicites proposés par *MOLAP* sont

pénalisant dans la phase de chargement de la base, tant en termes de performance que de volume.

Les systèmes *MOLAP* apparaissent comme une solution acceptable pour le stockage et l'analyse d'un entrepôt lorsque la quantité estimée des données ne dépasse pas quelques giga-octets et lorsque le modèle multidimensionnel évolue peu. Cependant, lorsque les données sont éparpillées, ces systèmes sont consommateurs d'espace [20] et des techniques de compression doivent être utilisées.

MOLAP est incompatible avec d'autres modes d'accès aux données. Si *MOLAP* doit cohabiter avec d'autres techniques d'accès aux données (par requête *SQL*, par data mining etc.), deux bases de données doivent cohabiter. En effet, *MOLAP* repose sur un moteur spécialisé, qui stocke les données dans un format tabulaire propriétaire (cube). Pour accéder aux données de ce cube, on ne peut pas utiliser le langage de requête standard *SQL*, il faut utiliser une *API* spécifique.

1.3.2 L'approche ROLAP

Les systèmes de type *ROLAP* utilisent un *SGBD* relationnel pour stocker et gérer les données de l'entrepôt. Ils fournissent pour le *SGBD* relationnel, une vision multi-dimensionnelle de l'entrepôt, des calculs de données dérivées et des agrégations à différents niveaux. Les systèmes de type *ROLAP* permettent la génération de requêtes *SQL* mieux adaptées au schéma relationnel grâce aux méta-données qui profitent des vues matérialisées existantes pour exécuter efficacement ces requêtes tout en diminuant sensiblement le coût lié à la mise en œuvre d'un serveur de base de données multidimensionnel supplémentaire. Pour cela, les outils *ROLAP* s'appuient pour la plupart sur une modélisation des données, distinguant les axes d'analyse et les faits à observer.

Deux schémas principaux sont utilisés pour modéliser les systèmes *ROLAP* : le schéma en étoile et le schéma en flocon de neige. Tous ces schémas distinguent les indicateurs ou mesures d'analyse (par exemple les quantités vendues) qui sont regroupés dans une table qu'on appelle la table des faits et les dimensions de l'analyse multidimensionnelle, que représentent un ensemble de tables de dimension (comme *Produit*, *Temps*, *Client*) représentant chacune un certain nombre de propriétés avec différents niveaux d'agrégation.

Les systèmes *ROLAP* peuvent stocker de grands volumes de données mais ils peuvent présenter un temps de réponse élevé [20] du fait qu'ils n'agrègent rien mais demandent de créer explicitement certains agrégats. De ce fait, ils sont plus faciles à intégrer dans les *SGBDs* relationnels existants mais plus lourds à administrer.

Pour optimiser les performances lors de la navigation dans les données ou pour les calculs complexes, les outils *ROLAP* proposent le plus souvent un composant serveur. Le tableau ci-dessous présente un certain nombre de différences entre le modèle *MOLAP* et *ROLAP*.

Approche	Avantages	Inconvénients
ROLAP	Standard bien établi Efficace pour le transactionnel Capacité d'expansion élevée (téra-octets)	Absence de vue conceptuelle SQL peut être inadéquat pour la formation de tableaux croisés Lent
MOLAP	Implémentation souvent plus performante que ROLAP Traitement de requête spécialisé Technique d'indexation spécialisée	Inadéquat pour le transactionnel Capacité d'expansion limitée (dizaines de giga-octets) Absence de standard

TABLE 1.3: Différences entre le modèle *MOLAP* et *ROLAP*

1.3.2.1 Modélisation en étoile

Le modèle en étoile est défini par une table relationnelle centrale appelée la table de faits. Cette table de faits se compose d'une liste d'attributs et de mesures.

Les attributs sont des clés étrangères permettant chacun de relier la table de faits à une des tables de dimension. La liste de mesures fournit des informations numériques qui présentent un intérêt pour l'analyse.

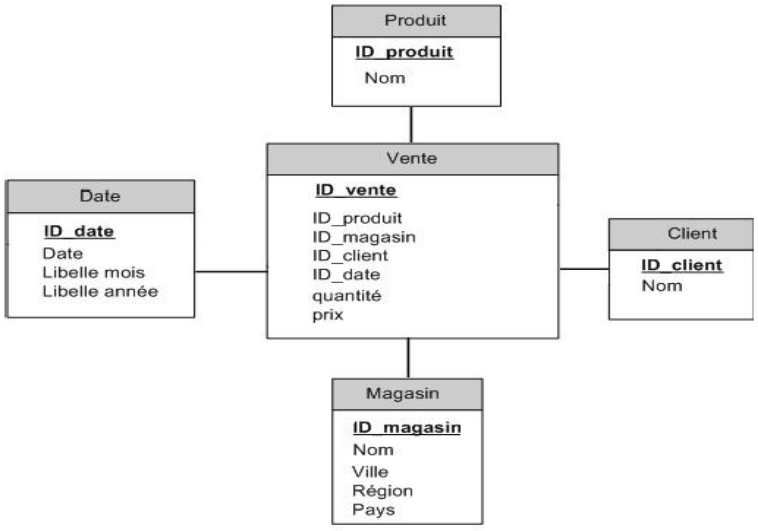

FIGURE 1.3: Schéma en étoile

La table de faits est normalisée et peut atteindre une taille importante par rapport au nombre de n-uplets. Les tables de dimension sont généralement dénormalisées afin de minimiser le nombre de jointures nécessaires pour évaluer une requête.

Le schéma en étoile est conçu pour répondre à des requêtes inhérentes à la structure dimension-fait. Les requêtes typiques de ce schéma sont appelées les requêtes de jointure en étoile qui ont les caractéristiques suivantes :

- Il y a des jointures multiples entre la table des faits et les tables de dimension.
- Il n'y a pas de jointure entre les tables de dimensions.

– Chaque table de dimension impliquée dans une opération de jointure a plusieurs prédicats de sélection sur ses attributs descriptifs.

Les entrepôts de données relationnels sont souvent représentés par un schéma en étoile [131] étant donné ces bonnes performances et la simplicité de sa mise en place.

1.3.2.2 Modélisation en flocon de neige

Le schéma en étoile ne reflète pas les hiérarchies associées à une dimension. Il exige que les informations complètes associées à une hiérarchie de dimension soient représentées dans une seule table, même lorsque les différents niveaux de la hiérarchie ont des propriétés différentes.

Pour résoudre ce problème, le schéma en flocon de neige est constitué d'une seule table de fait centrale et différentes tables de dimensions. Toute dimension à hiérarchies multiples est éclatée en au moins une sous table qui est gérée dans une structure séparée des autres hiérarchies.

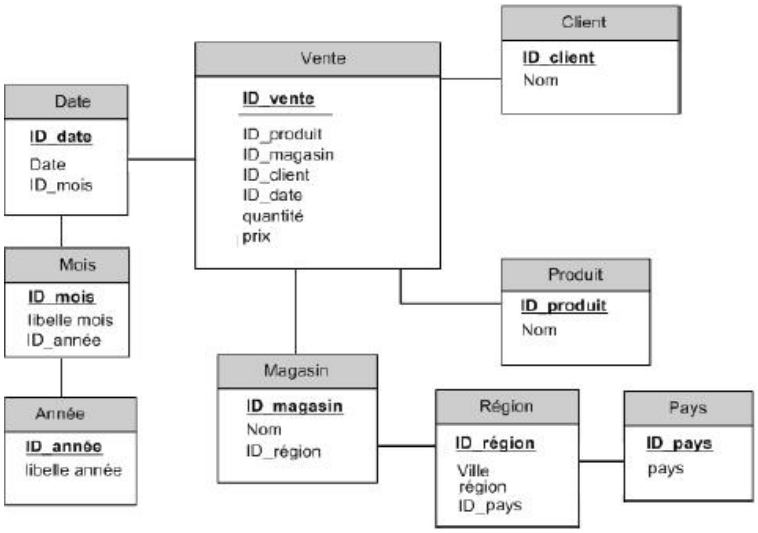

FIGURE 1.4: Schéma en flocon de neige

Ainsi, le schéma en flocons de neige de par sa structure permet de normaliser les dimensions et facilite l'analyse par sa représentation plus explicite de la hiérarchie entre les attributs.

Cependant, cette normalisation rend plus complexe la lisibilité et la gestion dans ce type de schéma. En effet, ce type de schéma augmente le nombre de jointures à réaliser dans l'exécution d'une requête.

1.3.2.3 Modélisation en constellation

Le schéma en constellation représente plusieurs tables de faits qui ont des tables de dimensions en commun. Cependant, chaque table de faits a ses propres dimensions [20]. Ainsi, pour chaque schéma en étoile, il est possible de construire un schéma constellation de fait. Ceci peut être obtenu en scindant le schéma en étoile d'origine en plusieurs schémas en étoile dont chacun décrit les faits à un autre niveau des hiérarchies de dimension.

Le principal défaut du schéma en constellation est sa conception plus complexe, car de nombreuses variantes de types d'agrégation particulières doivent être examinées et sélectionnées.

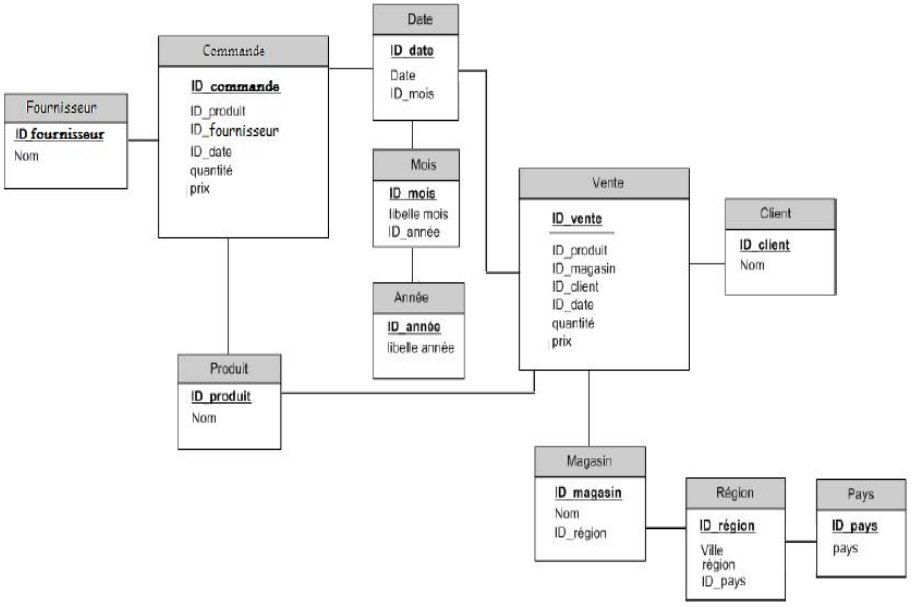

FIGURE 1.5: Schéma en constellation

La figure 1.5 montre un schéma en constellation qui est composé de deux tables de faits. La première s'appelle Commande et gère les différents produits commandés aux fournisseurs pendant un certain temps. La deuxième table s'appelle Vente et enregistre les quantités de produits qui ont été vendus dans les différents magasins pendant un certain temps.

1.4 Techniques d'optimisation de l'exploitation de l'entrepôt de données

Plusieurs structures ont été proposées pour optimiser les performances des entrepôts de données relationnels [222]. Ces dernières peuvent être classées en deux catégories : (1) les structures d'optimisation non redondantes et (2) les structures d'optimisation redondantes.

Dans la première catégorie, nous pouvons citer la fragmentation horizontale et le traitement parallèle. Dans la seconde catégorie, nous pouvons citer les index et les vues matérialisées.

1.4.1 La fragmentation

La fragmentation est une technique qui consiste, dans le cas des bases de données relationnelles, à partitionner une table relationnelle en plusieurs fragments (sous-ensembles de la table) qui peuvent être non disjoints. Une caractéristique intéressante liée à la fragmentation est le fait qu'elle ne duplique pas les données et par conséquent, elle réduit

les coûts de stockage et de maintenance. Deux sortes de fragmentation sont possibles : la fragmentation horizontale et la fragmentation verticale.

Dans la fragmentation verticale [186], [154] la table est partitionnée en fragments verticaux qui sont des projections appliquées à la table en fonction des attributs utilisés dans le processus de fragmentation. Cette technique optimise l'exécution des requêtes de projection mais la reconstruction des tables fragmentées verticalement nécessite l'opération de jointure, qui est très coûteuse.

La fragmentation horizontale [186] consiste à partitionner une table relationnelle en sous ensembles de n-uplets appelés fragments horizontaux. Cette opération peut s'effectuer soit grâce à des prédicats de sélection définis sur la table et dans ce cas on parle de fragmentation primaire ou bien elle s'effectue avec des prédicats de sélection définis sur une autre table qui a une jointure liant ces deux tables et dans ce cas on parle de fragmentation horizontale dérivée.

L'utilisation de la fragmentation verticale et horizontale dans les entrepôts de données permet d'optimiser l'exécution des requêtes en évitant le balayage des grandes tables. Dans ce contexte, deux cas d'utilisation de la fragmentation sont considérés. Dans le premier cas, la table de faits est partitionnée horizontalement en utilisant la fragmentation dérivée en fonction d'une ou plusieurs tables de dimension. Dans le deuxième cas, la table de faits est fragmentée verticalement en utilisant toutes les clés étrangères de la table des faits.

1.4.1.1 Problème de sélection d'un schéma de fragmentation

Le problème de sélection d'un schéma de fragmentation pour un entrepôt de données relationnel consiste à trouver la meilleure manière pour décomposer les tables de dimension, pour ensuite partitionner la table de faits. Etant donné que ce type de fragmentation de la table des faits pourrait engendrer un nombre important de fragments ce qui rendrait le processus de maintenance très coûteux.

1.4.2 La matérialisation des vues

Contrairement à une vue classique qui est calculée à chaque consultation à partir de la base de données, une vue matérialisée est définie comme une table contenant les résultats d'une requête. Les vues améliorent l'exécution des requêtes en pré calculant les opérations les plus coûteuses comme la jointure et l'agrégation et en stockant leurs résultats dans une table [108]. En conséquence, certaines requêtes nécessitent seulement l'accès aux vues matérialisées et sont ainsi exécutées plus rapidement [20].

Deux thèmes principaux ont caractérisé les nombreux travaux traitant des problématiques concernant les vues matérialisées dans le contexte des entrepôts.

1.4.2.1 La maintenance des vues matérialisées

Étant donné que les vues matérialisées doivent être en synchronisation avec les données source, toute modification apportée à la source doit être répercutée vers les vues.

La maintenance des vues matérialisées se propose donc de répercuter les mises à jour survenues au niveau des sources de données, afin d'assurer leur cohérence.

Les techniques de maintenance proposées peuvent être incrémentales. Dans ce cas les mises à jour de la source sont répercutées immédiatement au niveau des vues. Certains travaux traitent de l'approche d'auto-maintenance qui consiste à maintenir la vue en limitant les accès aux sources, notamment en utilisant des vues auxiliaires.

1.4.2.2 La sélection des vues matérialisées

Ce problème consiste à déterminer un ensemble optimal de vues à matérialiser permettant l'amélioration du temps de réponse en considérant des contraintes liées à l'espace de stockage des vues ou à leurs coûts de maintenance. Nous détaillerons plus ce problème dans les sections suivantes.

1.4.2.3 Réécriture automatique des requêtes

L'un des principaux avantages lors de la création des vues matérialisées, est la capacité de l'optimiseur de coût de réécrire une requête. L'optimiseur transforme une requête *SQL* basé sur des tables ou vues en une requête qui accède à une ou plusieurs vues matérialisées. Cette transformation est transparente pour l'application ou l'utilisateur final [167].

Avant que la requête soit réécrite, celle-ci est sujette à plusieurs contrôles pour déterminer si elle est candidate à la réécriture. Pour que la réécriture de la requête soit possible, il faut que :

- La base de données le permette (spfile)
- La vue matérialisée soit candidate. (Clause ENABLE QUERY REWITE).
- L'optimiseur soit autorisé à réécrire la requête (CHOOSE ou COST)
- Le client dispose des droits systèmes de réécriture

1.4.3 L'indexation

Un certain nombre de stratégies d'indexation ont été proposées pour les entrepôts de données. Ces dernières peuvent être classées en deux catégories: (a) les index défini sur un ou plusieurs attributs d'une même table (mono index) et (b) les index défini sur plusieurs tables (multi index).

1.4.3.1 Index en B-arbre

Un B-arbre est une liste chaînée de nœuds dont la valeur est celle de l'index. Si cet index est construit sur un attribut clé, les feuilles de l'arbre font référence à une seule valeur. Dans le cas où l'index est construit sur un attribut non-clé des n-uplets de la table indexée, les feuilles de l'arbre font référence à plusieurs valeurs. Cette référence spécifie l'emplacement physique du n-uplet sur le disque. Ainsi, le B-arbre est une structure d'index qui offre un excellent compromis pour les opérations de recherche par clé et par intervalle car sa mise à jour est dynamique. Ces qualités expliquent qu'il soit ainsi que ces variantes systématiquement intégrées dans la plupart des *SGBD* relationnelles [180].

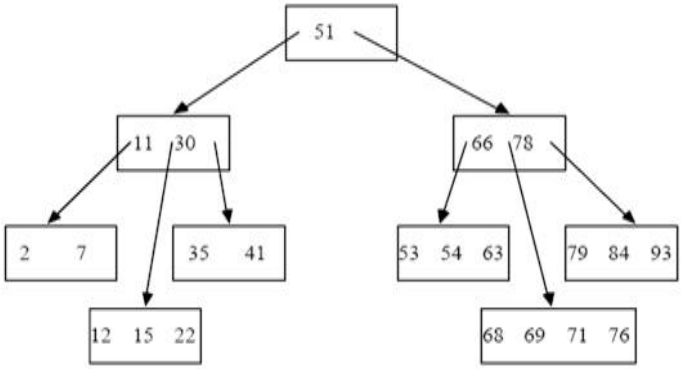

FIGURE 1.6: Schéma d'un index B-arbre d'ordre 2

1.4.3.2 Index sur liste de valeurs

Un value-list index se compose de deux parties. La première partie est une structure d'arbre équilibrée qui est généralement un B-Arbre avec des pourcentages d'utilisation. *Oracle* fournit un B*-arbre dont l'utilisation peut aller à 100%.

La deuxième partie est un schéma de correspondance. Ce schéma est attaché aux feuilles de l'arbre et il pointe vers les n-uplets de la table à indexer.

Selon la cardinalité de l'attribut indexé, deux types de structures d'index sont utilisées dans la deuxième partie (les nœuds feuilles du B-Arbre). Si la cardinalité de l'attribut indexé est relativement petite, la structure utilisée au niveau feuille du B-Arbre est un index binaire [165] représenté sous forme d'un vecteur de bits. Dans ce vecteur, chaque n-uplet d'une relation, est associé à un bit qui prend la valeur 1 si le n-uplet est membre de la liste ou 0 dans le cas contraire.

Si la cardinalité de l'attribut indexé est grande, une liste de *RowID* (Row Identifier) est associée à chaque valeur unique de la clé de recherche. Cette liste est partitionnée en un certain nombre de block disque chainés entre eux.

La figure 1.7 illustre un exemple d'un index de type Value List de 12 enregistrements avec structure en vecteurs bitmap.

	$\pi_A(R)$	B^8	B^7	B^6	B^5	B^4	B^3	B^2	B^1	B^0
1	3	0	0	0	0	0	1	0	0	0
2	2	0	0	0	0	0	0	1	0	0
3	1	0	0	0	0	0	0	0	1	0
4	2	0	0	0	0	0	0	1	0	0
5	8	1	0	0	0	0	0	0	0	0
6	2	0	0	0	0	0	0	1	0	0
7	2	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	0	1
9	7	0	1	0	0	0	0	0	0	0
10	5	0	0	0	1	0	0	0	0	0
11	6	0	0	1	0	0	0	0	0	0
12	4	0	0	0	0	1	0	0	0	0

(a) (b)

FIGURE 1.7: Exemple d'index sur liste de valeurs

1.4.3.3 Index de projection

Un index de projection est équivalent à la colonne en cours d'indexation. Si C est la colonne en cours d'indexation, l'index de projection sur C est constitué d'une séquence de valeurs de la colonne C, stockée dans le même ordre que le numéro ordinale de ligne de la table d'où les valeurs sont extraites (voir Figure 1.8).

Il a été montré dans [165] que les index de projection améliorent les autres schémas d'indexation pour la bonne exécution des requêtes impliquant des calculs sur deux ou plusieurs valeurs de la colonne et semblent améliorer passablement les requêtes de groupement (*Group by*).

Client

CID	CID	Nom	Age	Ville
1	616	Gilles	20	Londres
2	515	Yves	42	Paris
3	414	Patrick	21	Tokyo
4	313	Didier	52	Tokyo
5	212	Eric	18	Londres
6	111	Pascal	17	Londres

Index de projection sur âge

Age
20
42
21
52
18
17

FIGURE 1.8: Exemple d'index de projection

1.4.3.4 Index binaire

Un index binaire ou bitmap, considère toutes les valeurs distinctes de l'attribut indexé, que la valeur soit présente ou non dans la table [165]. Pour chacune de ces valeurs, nous construisons un bitmap (un tableau de bits), qui contient autant de bits que de tuples présents dans la table à laquelle l'attribut indexé appartient [224]. Le bitmap est codé avec une valeur de zéro ou un qui représente respectivement l'absence ou la présence de l'attribut à la position du bit correspondant au tuple.

L'index bitmap, étant codé sur des bits, il occupe moins d'espace qu'un *B-arbre* construit sur le même attribut particulièrement pour les attributs ayant une faible cardinalité. Ces caractéristiques lui confèrent des performances intéressantes dans le cadre d'indexation des requêtes de sélection dans le contexte des entrepôts de données gérant de gros volumes de donnée interrogées par des requêtes essentiellement de sélection classant les résultats par des attributs catégoriels définis sur de petits domaines de valeur [180]. Cependant, sa mise à jour étant assez coûteuse, il est alors moins performant dans un contexte où les données sont souvent modifiées [51], [223].

Client

CID	CID	Nom	Age	Ville
1	616	Omar	20	Annaba
2	515	Ali	42	Alger
3	414	Mourad	21	Oran
4	313	Yacine	52	Oran
5	212	Hamid	18	Annaba
6	111	Younes	17	Annaba

Index binaire ville

Annaba	Oran	Alger
1	0	0
0	0	1
0	1	0
0	1	0
1	0	0
1	0	0

FIGURE 1.9: Exemple d'index binaire

1.4.3.5 Index de jointure

La jointure est considérée comme l'une des opérations les plus couteuses. Ainsi, plusieurs structures ont été proposées pour optimiser les opérations de jointure. Ces dernières peuvent être classées en deux catégories. Dans la première catégorie, nous pouvons citer les différentes implémentations de l'opération de jointure : les boucles imbriquées, le tri fusion et le hachage. Cependant, ces techniques ne sont efficaces que lorsque la jointure concerne deux tables.

Dans le cas d'un contexte multi tables, Valduriez [204] a préconisé l'utilisation d'un index de jointure qui peut être vu comme une jointure pré-calculée, créé à l'avance. Il est implémenté par une relation d'arité 2, matérialisant les liens entre deux relations par le biais d'une table à deux colonnes, contenant les *RID* (identifiant de n-uplet) joints deux par deux (voir figure 1.10).

Client

RID^C	CID	Nom	Age	Ville
1	616	Omar	20	Alger
2	515	Ali	42	Oran
3	414	Mourad	21	Alger
4	313	Yacine	52	Oran

Magasin

RID^M	PID	Nom	Ville
1	101	Printemps	Oran
2	212	Dunes	Alger

Index de jointure

RID^S	CID	PID
1	616	212
2	414	212
3	515	101
4	313	101

FIGURE 1.10: Exemple d'index de jointure

Ce genre d'index est souhaité pour les requêtes *OLTP*, car elles possèdent souvent des jointures entre deux tables [204], [197]. Son efficacité dépend du coefficient de sélectivité de jointure. Si la jointure a une forte sélectivité, l'index de jointure sera petit et aura une grande efficacité. Cependant, pour les entrepôts de données modélisés par un schéma en étoile son efficacité reste limitée étant donné le nombre important d'opérations de jointu-

res (entre la table des faits et plusieurs tables de dimension) qui caractérise ces requêtes décisionnelles.

Index de jointure en étoile Dans le contexte des entrepôts de données, Red Brick [197] a proposé un nouvel index de jointure, appelé, index de jointure en étoile (star join index), adapté aux requêtes de jointure en étoile comportant généralement plusieurs jointures entre la table des faits et les tables de dimension.

L'index de jointure en étoile peut contenir toutes les combinaisons possibles entre l'identifiant de la table des faits et les clés étrangères des tables de dimension. Cependant, la construction d'un index permettant de joindre toutes les tables de dimension avec la table des faits reste difficile étant donné le coût de stockage et de maintenance important qu'il nécessite. Ainsi, un index partiel est construit en joignant certaines tables de dimension avec la table de faits.

Index de jointure binaire L'index de jointure binaire est une variante de l'index de jointure en étoile. Etant donné que les requêtes de jointure en étoile possèdent des opérations de jointure suivies par des opérations de sélection, un bitmap représentant les n-uplets de la table de faits est créée pour chaque valeur distincte de l'attribut de la table dimension sur lequel l'index est construit. Le ième bit du bitmap est égal à 1 si le n-uplet correspondant à la valeur de l'attribut indexé peut être joint avec le n-uplet de rang i de la table de faits. Dans le cas contraire, le ième bit est à zéro (voir figure 1.11).

Client

RID^C	CID	Nom	Age	Ville
1	616	Gilles	20	Londres
2	515	Yves	42	Paris
3	414	Patrick	21	Tokyo
4	313	Didier	52	Tokyo
5	212	Eric	18	Londres
6	111	Pascal	17	Londres

Produit

RID	PID	Nom	Gamme
1		Sonoflore	Beauté
2		Clarins	Beauté
3		WebCam	Multimedia
4		Barbie	Jouets
5		Manure	Gardening
6		SlimForme	Sport

Temps

RID	TID	Mois	Année
1	11	Janvier	2003
2	22	Février	2003
3	33	Mars	2003
4	44	Avril	2003
5	55	Mai	2003
6	66	Juin	2003

Vente

RID^S	CID	PID	TID	Montant
1	616	106	11	25
2	616	106	66	28
3	616	104	33	50
4	515	104	11	10
5	414	105	66	14
6	212	106	55	14
7	111	101	44	20
8	111	101	33	27
9	212	101	11	100
10	313	102	11	200
11	414	102	11	102
12	414	102	55	103
13	515	102	66	100
14	515	103	55	17
15	212	103	44	45
16	111	105	66	44
17	212	104	66	40
18	515	104	22	20
19	616	104	22	20
20	616	104	55	20
21	212	105	11	10
22	212	105	44	10
23	212	105	55	18
24	212	106	11	18

Index de jointure binaire sur ville

RID^S	Londres	Paris	Tokyo
1	1	0	0
2	1	0	0
3	1	0	0
4	0	1	0
5	0	0	1
6	1	0	0
7	1	0	0
8	1	0	0
9	1	0	0
10	0	0	1
11	0	0	1
12	0	0	1
13	0	1	0
14	0	1	0
15	1	0	0
16	1	0	0
17	1	0	0
18	0	1	0
19	1	0	0
20	1	0	0
21	1	0	0
22	1	0	0
23	1	0	0
24	1	0	0

FIGURE 1.11: Exemple d'index de jointure binaire

Les index de jointure binaires sont efficaces pour les requêtes de type *COUNT*, *AND*, *OR*, *NOT*, d'où leur implémentation dans les *SGBD*s commerciaux, comme *Oracle 10g*. Notons que les index de jointure binaires sont particulièrement recommandés pour des attributs ayant une faible cardinalité comme genre. Notons aussi que dans un contexte décisionnel, les requêtes d'analyse se font généralement sur des indicateurs (attributs) dont le domaine est restreint.

Une autre caractéristique offerte par les index de jointure binaires est la compression [126], d'où une réduction de leur espace de stockage et la possibilité de les stocker en mémoire centrale.

Les index de jointure binaires peuvent être définis sur un seul ou plusieurs attributs non clés des tables de dimension, comme le montre l'exemple suivant :

CREATE BITMAP INDEX sales_c_gender_bjix

ON sales(customers.cust_gender)

FROM sales, customers

WHERE sales.cust_id = customers.cust_id

Cette instruction qui crée un index de jointure binaire entre la table des faits *sales* et la table de dimension *customers* sur l'attribut de sélection *gender*. Le résultat de cet index est une table avec trois colonnes, représentant respectivement, le numéro de ligne de la table des faits (Row Identifier) participant dans la jointure, une colonne binaire pour la valeur de sexe féminin et une colonne binaire pour la valeur de sexe masculin.

Supposons que nous ayons la requête suivante : *SELECT* count(*) *FROM* sales, customers *WHERE* gender='F' *and* sales.cust_id = customers.cust_id. L'index de jointure sales_c_gender_bjix peut être utilisé pour accélérer cette requête, en utilisant le "hint" suivant :

SELECT /*+ INDEX(sales_c_gender_bjix)*/ count(*)

FROM sales, customers

WHERE gender='F' and sales.cust_id = customers.cust_id.

Dans ce cas, la requête utilise uniquement cet index et sans aucun accès à la table des faits, ce qui représente un gain considérable (on économise une opération de jointure).

1.5 Problème de sélection d'index

Dans cette section, on formalise d'abord le problème de sélection d'index. Par la suite, nous présentons les plus importantes approches s'intéressant à la résolution de ce problème.

1.5.1 Formalisation du problème de sélection d'index

Le problème de sélection d'un ensemble d'index optimal pour une base de données a été étudié depuis les années 70. Il consiste à sélectionner, à partir d'un ensemble de requêtes décisionnelles et la contrainte d'une ressource donnée (l'espace stockage, le temps de maintenance etc.), un ensemble d'index afin de minimiser le coût d'exécution des requêtes. Formellement, le problème de sélection des index est formulé de la façon suivante :

étant donné : un entrepôt de données modélisé par un schéma en étoile formé d'une table des faits F et de $D = \{D_1, D_2, ...D_d\}$ tables de dimensions, un ensemble de requêtes d'interrogation $Q = \{q_1, q_2, ..., q_m\}$ (les plus fréquentes) et une contrainte de stockage S.

L'objectif du problème de sélection des index de jointure binaires est de sélectionner un ensemble d'index $I = \{I_1, I_2, ..., I_p\}$ réduisant le coût d'exécution de requêtes et respectant la contrainte de l'espace de stockage S.

1.5.2 Complexité de la sélection d'index

La sélection d'index est un problème difficile [42], [20] étant donné la grande volumétrie du schéma d'un entrepôt de données relationnel (plusieurs tables avec plusieurs colonnes) et la complexité des requêtes d'interrogation de type *OLAP*.

Par conséquent, l'espace de recherche qui se rapporte à une charge de requêtes peut être très grand étant donné le nombre important des attributs candidats participant à la construction des index [205].

1.5.3 Approches de sélection d'index

Plusieurs approches ont été proposées dans la littérature pour résoudre le problème de sélection d'index. La plupart de ces approches utilisent deux phases : (1) la sélection des index candidats et (2) la sélection d'une configuration finale d'index.

1.5.3.1 Phase de sélection d'index candidats

Cette phase permet de réduire l'espace de recherche du problème de sélection d'index en éliminant les attributs non pertinents. Pour élaguer l'espace de recherche des index, de nombreuses approches ont été proposées [9], [7], [54], [52], [138], [205].

Ces approches peuvent être classées en deux catégories: (1) les approches heuristiques et les approches axées sur le data mining.

Approches heuristiques Dans cette catégorie, des méthodes heuristiques sont utilisées pour obtenir une configuration d'index appropriée et approcher au mieux de la solution optimale.

Par exemple, dans [54], un algorithme glouton est proposé. Il utilise le coût de l'optimiseur de *SQL* Server pour décider de la qualité d'une configuration donnée d'index. La faiblesse de cette approche est qu'elle impose le nombre de candidats générés.

DB2 Advisor est un autre exemple appartenant à cette catégorie [205]. Dans *DB2 Advisor* l'analyseur de requêtes est utilisé pour ramasser les attributs de sélection utilisés

dans la charge de requêtes. Les candidats générés sont obtenus en effectuant quelques combinaisons des attributs de sélection [205].

Approches data mining Dans les approches data mining, le processus d'élagage est fait en utilisant des techniques data mining. Par exemple, dans [9], [7], [12], respectivement l'algorithme *Close* [172] et *GenMax* [97] sont utilisés pour générer des index candidats, sans imposer leur nombre comme dans la première catégorie. L'idée de base de ces approches est que les attributs qui apparaissent fréquemment ensemble dans les requêtes constituent de bons index candidats.

1.5.3.2 Phase de sélection d'index finaux

Dans cette phase, la construction de la configuration finale d'index est le plus souvent assurée par une démarche ascendante ou descendante à l'aide d'algorithmes calculant la performance des index sélectionnés tel que les algorithmes gloutons [54] ou les algorithmes de programmation linéaire [52] etc.

Dans une démarche ascendante la configuration d'index finale est considérée au départ comme vide. Par la suite des index candidats réduisant le plus le coût d'exécution de requêtes y sont ajoutés itérativement jusqu'a ce qu'il n'y ait plus d'amélioration [205], [54], [205], [96].

Au contraire, la démarche descendante démarre avec l'ensemble complet des index candidats. A chaque itération, des index sont élagués jusqu'à ce que le coût de la charge ne diminue plus [85].

Enfin, certaines démarches combinent les deux approches [56], [105]. Les algorithmes utilisés dans cette phase de sélection sont basés sur des modèles de coût calculant la performance des index sélectionnés.

Deux types de modèles sont disponibles : (1) des modèles de coût de l'optimiseur de requêtes du *SGBD* utilisé, comme dans les travaux du groupe base de données de *Microsoft* [205], [54] et les travaux du groupe de bases de données d'*IBM* pour *DB2 advisor* [178] et (2) des modèles de coût mathématique utilisés pour valider les algorithmes proposés [56] ,[105], [105], [73].

1.5.4 Travaux sur la sélection d'index

Nous présentons ici un certain nombre de travaux qui nous paraissent très important dans la problématique de sélection d'index sans toutefois prétendre être complet.

1.5.4.1 Travaux de Frank et al

Ces travaux, menés à l'Université de *Georgia Tech*, avaient pour but de proposer un outil d'aide à la décision pour le choix d'index pour une base de données [85].

La démarche de sélection se base sur un dialogue établi entre l'outil de sélection d'index et l'optimiseur de requêtes afin de calculer le gain de performance qu'apporte l'utilisation d'un index.

Le gain est représenté sous forme d'un graphe d'index. Il est calculé par la formule suivante :

Gain = Coût du parcours séquentiel – Coût avec index.

L'estimation du coût d'un index se fait par l'optimiseur de requêtes de la façon suivante : Pour chaque requête q de la charge, un ensemble d'index candidat est généré par l'optimiseur qui donne selon un modèle son coût d'utilisation.

De nouveaux ensembles d'index sont générés et les gains de performance de chaque index sont sommés et ceux présentant un gain total positif sont proposés à l'utilisateur.

1.5.4.2 Travaux de Whang - Algorithmes ADD and DROP

Whang et al, [213] ont proposé une approche ascendante et une autre descendante pour la sélection d'index. L'approche ascendante qui est implémentée par l'algorithme de sélection *DROP*, commence d'abord en considérant tous les index possibles.

Durant chaque itération, l'index engendrant la plus grande décroissance du coût des requêtes est éliminé. Quand cette réduction du coût n'est plus possible en éliminant un seul index, l'algorithme *DROP* élimine deux index à la fois, trois index à la fois et ainsi de suite, jusqu'à ce que ceci ne soit plus possible.

L'approche ascendante qui est implémentée par l'algorithme de sélection *ADD*, initialise le processus d'optimisation par l'ensemble vide (aucun index n'est encore sélectionné). A chaque itération, l'algorithme *ADD* ajoute un index réduisant le coût d'exécution des requêtes. Le processus s'arrête lorsqu'aucune réduction de coût n'est possible.

1.5.4.3 Travaux de Brunel, Rollin et al,.

Ces travaux ont été effectués dans le cadre d'une collaboration entre l'université d'Oklahoma et l'université Lyon 2. Deux outils *IUS* (Index Usage Statistics) et *ISCA* (Index Selection and Creation Algorithm) ont été réalisés pour la sélection automatique des index [40].

a) L'outil IUS (Index Usage Statistics)

IUS qui est inspiré des travaux de Frank et al., [85] dialogue avec l'optimiseur de requêtes mais évite la construction du graphe d'index et le calcul des gains de performances.

Cet outil facilite le choix des index à construire en établissant un ensemble de statistiques représentant l'ordre d'importance de chaque index de l'ensemble initial pour chacune des requêtes de la charge. Ces statistiques sont obtenues, à partir d'un ensemble d'index, par l'optimiseur qui effectue un travail itératif couvrant toutes les requêtes de la charge.

A chaque itération et tant qu'il y a des index utiles, l'optimiseur propose pour la requête courante le meilleur index (moins coûteux) à utiliser et l'outil lui affecte un rang. Par la suite, on enregistre les index et leurs rangs trouvés pour la requête traitée et on réinitialise l'ensemble d'index à l'ensemble initial.

A la fin, on obtient un ensemble de statistiques représentant l'ordre d'importance de chaque index de l'ensemble initial dans chacune des requêtes de la charge. Cela facilite à l'administrateur le choix des index à construire.

b) L'outil ISCA (Index Selection and Creation Algorithm)

A partir d'une charge représentant uniquement l'ensemble des requêtes de sélection obtenues à partir du fichier l*og*. *ISCA* extrait les attributs présents dans chaque requête et leur affecte à chacun un identificateur qui est un numéro binaire établi dans sa table d'appartenance.

ISCA calcule pour chaque attribut de la table, le nombre de fois (fréquence) où il a été utilisé dans l'ensemble des requêtes de la charge et aussi le nombre de fois où il a été utilisé avec un ou plusieurs autres attributs extraits. Ensuite, l'administrateur de la base de données fixe un seuil et ne conserve que les attributs dépassant ce seuil.

1.5.4.4 Travaux de Chaudhuri et al

Le groupe base de données de Microsoft a développé l'outil de sélection d'index *IST* (Index Selection Tool) [53] qui permet, suivant des contraintes spécifier par l'utilisateur, de sélectionner d'une manière automatique des index pour une charge constituée d'un ensemble de requêtes *SQL*.

L'outil prend un ensemble de requêtes définies sur un schéma de base de données et réalise grâce à trois modules, les traitements suivants :

a) Sélection des index candidats

Pour chaque requête de la charge, *IST* considère comme des index candidats de départ l'ensemble des attributs présents dans une des clauses : *WHERE, GROUP BY, ORDER BY* et les attributs mis à jour dans un *UPDATE*.

Ces derniers permettent la construction d'une configuration d'index grâce aux deux modules : cost-evaluation et What-if index creation [55].

La meilleure configuration est retenue. L'ensemble d'index résultat est l'union des configurations obtenues pour chaque requête.

b) Elagage des configurations d'index

À partir des n index candidats de l'étape précédente, les k meilleurs sont sélectionnés à l'aide d'un algorithme glouton. Ce module est en dialogue permanent avec l'optimiseur et suit les étapes ci-dessous :

1. initialisation avec la meilleure configuration de taille m petite (m<<k);
2. augmentation de la configuration courante avec l'index minimisant le coût;
3. tant que le coût diminue et que le nombre des k index n'est pas atteint, revenir à l'étape 2.

Le résultat est un ensemble d'index constitué de l'union des configurations obtenues après élagage.

b) Génération des configurations d'index multi-attributs

Les index obtenus après élagage sont mono-attribut. Pour construire les index multi-attributs, IST utilise ces index mono-attribut et procède par augmentation grâce à deux démarches : MC-LEAD, MC-ALL. MC-LEAD : Un index candidat est un index mono-attribut combiné avec un attribut indexable. MC-ALL : Un index candidat est donc une combinaison d'index mono-attributs. Le même travail est effectué pour construire les d'index de taille supérieure à 2.

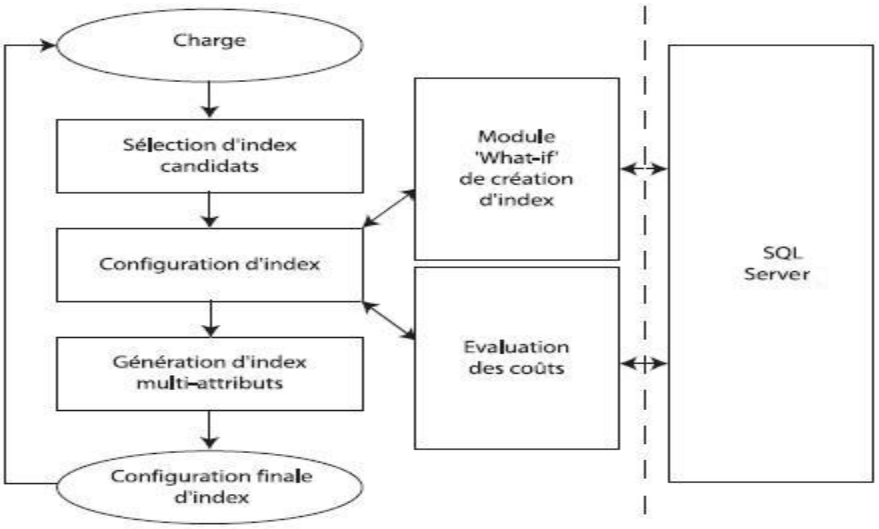

FIGURE 1.12: Architecture générale d'IST [7]

1.5.4.5 Travaux de Kratica et al

Dans le contexte de base de données, Kratica et al., [135] proposent l'utilisation d'un algorithme génétique pour sélectionner une configuration d'index qui minimise le temps de réponse d'une charge de requête donnée.

Pour cela, ils utilisent une formulation mathématique pour évaluer le gain en temps d'exécution et le coût de maintenance de chaque index candidat. L'objectif est de construire une configuration d'index de telle sorte que le temps total d'exécution de toutes les requêtes soit minimisé.

L'utilisation de l'algorithme génétique permet de déterminer la population d'individus élu qui peut survivre à la prochaine génération. Puisque la valeur objective évaluée est le temps de consommation d'opération, certaines valeurs déjà calculées sont sauvegardées dans une table en cache afin d'éviter de recalculer la valeur objective de certains individus.

Les valeurs de gain sont sauvegardées de manière efficace dans une matrice dont les lignes représentent les requêtes et les colonnes représentent les configurations possibles. Seules les valeurs de gain supérieurs à zéro sont sauvegardées. Chaque solution S est encodée par un vecteur binaire. Selon les valeurs des index construit, les configurations d'index actives sont déterminées.

Pour chaque requête i, il faut trouver la configuration $k*$ dont le gain en temps est minimisé. Finalement, il faut sommer des gains de toutes les requêtes et puis soustraire le temps de maintenance des index construits.

1.5.4.6 Travaux de Feldman et al

Feldman et al., [73] ont développé *DINNER,* un outil à base de connaissances qui assiste l'administrateur dans la sélection d'index.

DINNER recommande une configuration d'index qui contient un index primaire et un ensemble d'index secondaires, en utilisant une base de connaissances extraite des infor-

mations tirées des différentes tables, de leurs statistiques, des requêtes d'interrogation, de l'expertise de l'administrateur de la base de données, du manuel du *SGBD* utilisé, des cours sur l'administration des bases de données etc.

1.5.4.7 Travaux de Valentin et al

Le système *DB2 Advisor*, dont l'architecture est présentée dans la figure 1.13, modélise le problème de sélection d'index comme une variante du problème de sac à dos [205].

DB2 Advisor permet de recommander un ensemble d'index [205] en se basant sur un composant de l'optimiseur de requête. Pour cela, ce système admet comme paramètres d'entrée une charge de requête *SQL* et des statistiques décrivant la base de données cible et en sortie l'ensemble des index recommandés.

Le système *DB2 Advisor* est composé de :

1. Un ensemble d'interfaces à partir des lesquelles l'administrateur peut respectivement, sélectionner ou saisir directement des requêtes, spécifier des contraintes sous lesquelles le processus de sélection d'index se déroule (espace de stockage, la durée d'exécution,..). Il faut noter que la charge de requête est stockée dans une table propre à l'utilisateur courant appelée *ADVISE_ WORKLOAD*.
2. Un outil qui permet de récupère des informations sur les requêtes : fréquence d'exécution (grâce à *SQL* Cache de *BD2* Universal Database (*DB2 UDB*), celles récemment exécutées ou compilées.
3. Un outil en ligne de commande qui peut être invoqué par des commandes ou par l'interface graphique. Il permet d'invoquer l'optimiseur de DB2 pour chaque requête de la table *ADVISE_ WORKLOAD* pour évaluer ou recommander des index.
4. Les index recommandés par l'optimiseur sont stockés, pour l'utilisateur actuel, dans une table appelée *ADVISE_ INDEX*.

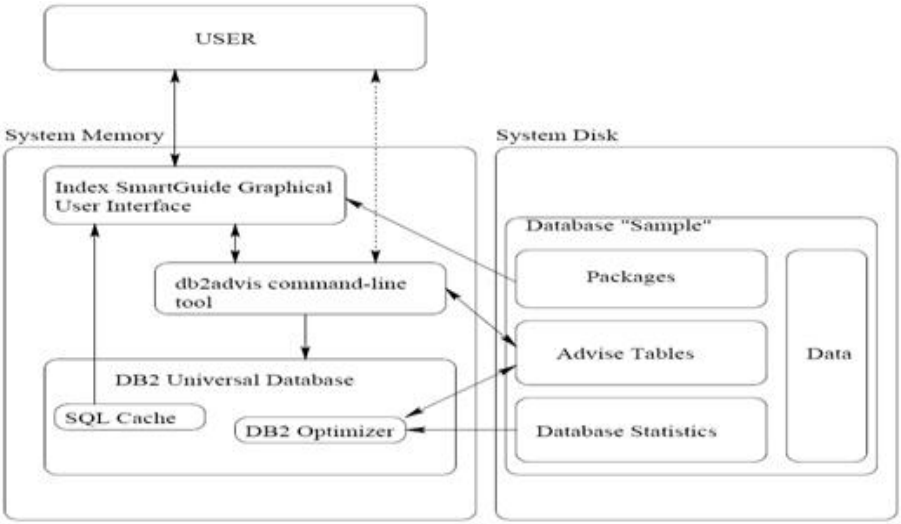

FIGURE 1.13: Architecture du système DB2 Advisor

Initialement, en se basant sur des connaissances relatives au *s*ystème de gestion de base de données utilisé et à l'analyse des clauses et prédicats des requêtes, l'optimiseur suggère un ensemble d'index virtuel pour chaque requête.

Par la suite, il utilise l'ensemble des index virtuels et des statistiques sur la base de données, pour générer des plans d'exécution pour chaque requête Il choisit ensuite, à partir de ces plans, le plus optimal d'entre eux pour le recommander comme index s'il contient un ou plusieurs index virtuel.

A partir des recommandations de l'optimiseur pour chaque requête, l'algorithme de sélection d'index est utilisé pour rechercher une combinaison optimale d'index pour toute la charge.

Pour remédier à certains problèmes décrits dans [205], l'outil utilise une routine qui crée une variante de chaque solution trouvée en permutant un petit nombre d'index appartenant à la solution pour un petit ensemble d'index qui n'est pas dans la solution. La charge de requête est ensuite réévaluée avec cette nouvelle configuration d'index virtuels. Si cette nouvelle solution est moins couteuse, elle devient la solution courante. La routine continue jusqu'à ce que le temps d'exécution maximum définit par l'utilisateur s'écoule.

1.5.4.8 Travaux de Golfarelli et al

Golfarelli et al., [96] ont proposé, dans le contexte des entrepôts de données modélisés en étoile, une approche heuristique pour résoudre le problème de sélection d'index de type liste de valeurs et bitmap. Sous la contrainte d'espace de stockage, l'approche préconise deux étapes pour la sélection d'index.

La première étape consiste à générer un ensemble d'index candidats. Ceci passe par la mise en place d'un modèle de coût mathématique permettant d'évaluer les plans d'exécution possibles des requêtes préalablement établis par un modèle de l'optimiseur de requêtes.

La deuxième étape utilise un algorithme glouton pour sélectionner, à partir de l'ensemble des candidats, la configuration optimale tout en respectant la contrainte d'espace de stockage. L'approche proposée reçoit en entrée les différents paramètres suivants :

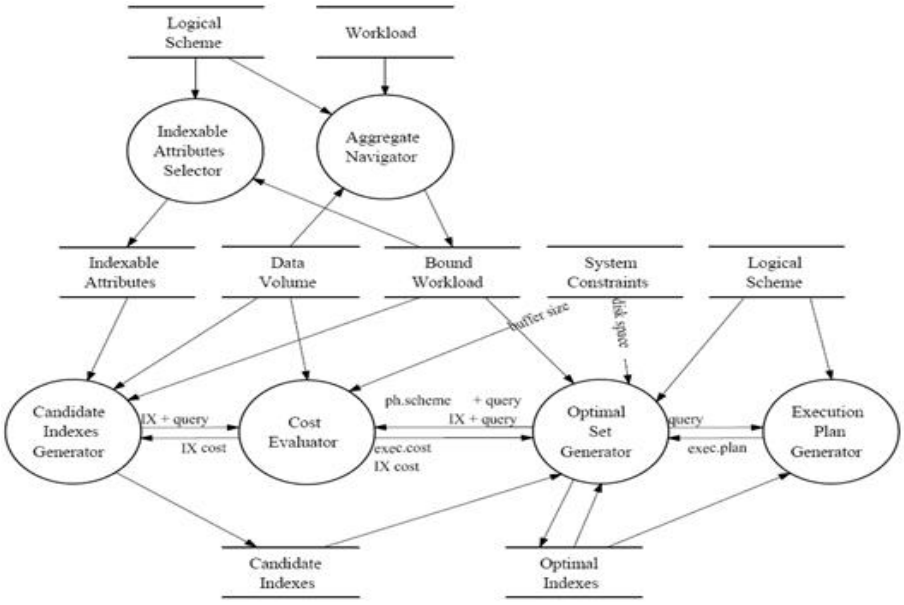

FIGURE 1.14: Architecture du système de sélection d'index

1. *DW Logical Scheme :* Contient des informations sur la structure des tables et vues, ainsi que le modèle logique.
2. Workload : Contient un ensemble de requêtes exécutées sur le DW avec leurs fréquences.
3. *Data Volume :* Contient des informations nécessaires pour évaluer le coût d'exécution des requêtes, par exemple : cardinalité des attributs.
4. *System Constraints :* Contient certaines contraintes imposées par le système, par exemple : l'espace de stockage maximum réservé aux index, taille du buffer. Les nœuds encerclés dans la figure précédente, représentent les fonctionnalités du système, voici un petit détail de chacune :

– *Aggregate Navigator :* étant donné une charge de requête et un schéma logique de l'entrepôt de données contenant une ou plusieurs vues matérialisées, Ce composant permet de sélectionner la meilleure vue avec laquelle chaque requête peut être résolue. La prise en considération des index n'est pas obligatoire.
– *Selector of indexable attributes :* ce module détermine, en se basant sur la structure des requêtes, les attributs utiles pour l'indexation des tables de dimension.
– *Generator of candidate indexes :* ce composant permet d'évaluer, pour chaque attribut indexable, le type d'index le plus adéquat. Les index sélectionnés par ce composant, présentés sous forme de couples (attribut, type d'index), constituent la liste des index candidats.
– *Optimal Set Generator :* ce composant permet de sélectionner la configuration d'index optimal incluant un sous-ensemble d'index candidats sur les attributs des tables dimensions (DT) ainsi que tous les index construits sur les attributs clés des DT et de la table de Faits (FT).

4. Cost Evaluator : ce module permet d'évaluer le coût d'accès pour chaque index.

5. Execution Plan Generator : étant donné un schéma physique, une requête et la table de faits sur laquelle la requête doit s'exécuter, ce module permet de générer le meilleur plan d'exécution qui résout la requête.

Chaque composant décrit ci-dessus, donne des résultats en sortie. Ces résultats peuvent être des entrées pour d'autres composants, comme ils peuvent être les solutions finales. Voici un petit résumé de chaque résultat en sortie :

- *Bound Workload :* ce résultat permet de stocker, pour chaque requête, une référence à la table de faits utilisée pour résoudre la requête.
- *Indexable Attributes :* contient l'ensemble de tous les attributs pouvant accélérer le temps de réponses des requêtes.
- *Candidate Indexes :* contient, pour chaque attribut indexable, le type d'index le plus approprié.
- *Optimal Indexes :* contient la configuration d'index optimal à créer.

1.5.4.9 Travaux de Dogac et al.

MAESTRO7 (Metu Automated indEx Selection Tool foR Oracle7) [66] est un outil de sélection automatique d'index spécifique pour *ORACLE* 7. Il a été proposé par le centre de recherche et développement de logiciel *TUBITAK* de l'université technique à *Ankara*, *Turquie*. Il permet de proposer, parmi un ensemble complet d'index, des index primaires (clustering index) et secondaires (non-clustering index) en tenant compte du coût de maintenance.

MAESTRO7 commence par extraire automatiquement, à partir du fichier *Log*, les requêtes *SQL* et leurs statistiques pour construire la charge de requête. Ensuite, il classifie les requêtes produisant le même plan d'exécution et met à jour leurs poids en les cumulant. Il détermine ensuite les colonnes susceptibles d'être des index dans le but d'améliorer les performances. Pour cela, il utilise la logique du *SGBD ORACLE7* dans l'utilisation ou non des index. Ainsi aucune colonne présentant ces caractéristiques n'est sujette à être candidate :

- Si une colonne est modifiée dans une clause *WHERE*
- Si une colonne est utilisée avec : *NOT IN, NOT LIKE, IS NOT NULL, IS NULL*
- Si elle est utilisée avec la clause LIKE pour comparer une chaine de caractère ayant le '%' au début.
- Les colonnes d'une même table apparaissant dans les deux cotes d'un opérateur de comparaison, car un parcours séquentiel doit être effectué dans cette situation.
- Si une requête contient une sous-requête, il est nécessaire d'exécuter la sous-requête pour chaque ligne de la requête. Donc l'index sur la colonne de la requête qui est compare au résultat de la sous-requête n'est pas utile.

Par la suite, il considère les colonnes utilisées avec l'une des clauses suivantes comme candidates pour les index de hachage :

$< nom\_colonne > = < constante > \big| < sous\text{-}requête >$

$< nom\_colonne > LIKE < constante >$

$< nom\_colonne > IN < liste\_de\_valeurs > \big| < sous\text{-}requête > .$

Cependant, étant donné qu'*ORACLE7* utilise un index de hachage statique qui n'est performant que s'il est utilisé sur une table statique. L'outil supprime, grâce aux informations déjà collectées, toutes les colonnes candidates pour l'index de hachage dont la table correspondante est considérée comme non statique. Une table est considérée comme non statique si la différence entre le nombre de lignes insérées et les lignes supprimées est supérieure au nombre de ligne total par un certain pourcentage. Le reste des colonnes indexables sont candidates pour l'index $B^{+}Tree$.

1.5.4.10 Travaux de Choenni et al

Dans le contexte des bases de données relationnelles, Choenni et al., [56] utilisent un modèle mathématique pour réduire l'ensemble des candidats pour le problème de sélection d'index.

Il se base sur le fait que l'ajout d'un index à une configuration peut augmenter son coût, le réduire, comme il peut aussi ne pas le changer. Les propriétés suivantes permettent de minimiser l'espace de recherche à explorer afin d'éviter d'effectuer une recherche exhaustive.

Les deux premières propriétés sont spécifiques à une fonction mathématiques supermodulaire et les deux dernières sont spécifiques à une fonction sous-modulaire. Il est à noter que les propriétés 1 et 4 sont équivalentes et les propriétés 2 et 3 sont équivalentes [7].

Proposition 1 : supposant un index i à ajouter à un ensemble d'index I. Si la valeur de C ne décroit pas suite à cet ajout, alors elle ne décroit pas non plus en ajoutant l'index i à tout ensemble I' contenant I. Cela signifie que l'index i n'appartient pas à la configuration d'index optimale.

Proposition 2 : supposant que l'index i soit à éliminer de l'ensemble I. Si la valeur de C n'est pas réduite suite à cette élimination, la valeur de C ne peut pas l'être pour tout index i éliminé de l'ensemble I' contenu dans I. Cela signifie que l'index i appartient à la configuration d'index optimale.

Proposition 3 : supposant que l'index i est avantageux pour un ensemble d'index I, il l'est aussi pour tout ensemble d'index I' contenant I. Cela signifie que i fait partie de la configuration optimale.

Proposition 4 : si un index i réduit le coût d'un ensemble d'index I. Suite à son élimination de cet ensemble, ce coût est aussi réduit pour tout ensemble I' contenu dans I. Cela signifie que i ne fait pas partie de la configuration optimale.

L'algorithme de sélection d'index prend en entrée une fonction de coût C, un ensemble d'attributs à indexer selon l'administrateur du système et une configuration initiale d'index pertinente pour chaque requête de la charge. Il considére aussi un sous-ensemble de Q notés Q_red. Q_red étant l'ensemble de requêtes de la charge Q pour lesquelles aucun index parmi la configuration initiale n'est avantageux.

L'algorithme renvoie, pour chaque sous-ensemble Q_red, un ensemble d'index avantageux constituant la configuration d'index optimale et un autre ensemble d'index désavantageux n'appartenant pas à la configuration d'index optimale.

L'évaluation des coûts des requêtes en présence de ces index mène généralement à décomposer chaque sous-ensemble de *Q_red* en deux groupes. Un groupe *G*1 contenant les requêtes rendant la fonction de coût *C* super-modulaire et un groupe *G2* contenant celles rendant la fonction de coût *C* sous-modulaire. A chaque groupe sont appliqués les propositions adéquates afin d'optimiser l'espace de recherche à explorer.

Enfin, une fusion des ensembles d'index avantageux et des ensembles d'index désavantageux est nécessaire afin de n'avoir qu'un seul ensemble d'index avantageux et un seul ensemble d'index désavantageux pour pouvoir résoudre le problème de sélection d'index avec une recherche exhaustive. Le temps et la faisabilité de cette recherche dépend du nombre d'index à engendrer.

1.5.4.11 Travaux de Gündem

La méthode présentée par Gündem [105], utilise un ensemble d'index candidats fourni par l'administrateur pour y sélectionner un sous-ensemble qui minimise, suivant une erreur tolérée, le coût de traitement des opérations de mise à jour et de sélection sans violer la contrainte d'espace de stockage.

Cette méthode prend en compte le fait qu'un attribut donné peut être indexé suivant plusieurs techniques et que, par conséquent, les index construits sur cet attribut suivant différentes techniques apportent des gains et occupent des espaces de stockage différents. En effet, un index candidat peut être un B-arbre, un index d'ordre, un ensemble d'index partiels etc. Ainsi, pour les attributs correspondants aux tables de la base considérée, l'ensemble des index associés à un attribut donné est appelé une classe d'équivalence et l'ensemble de ces classes constitue une partition. Le processus de sélection est subdivisé en deux étapes.

La première étape consiste à effectuer une optimisation locale pour calculer un ensemble disjoint d'index candidats qui maximisent le gain.

En premier, on défini un seul index candidat pour chaque attribut (un index par classe d'équivalence). Par la suite, on calcule, grâce à une fonction de coût. Le gain pour chaque index candidat est calculé en évaluant son coût de construction et l'apport obtenu par son utilisation pour les opérations de sélection et de mise à jour. Le résultat de cette étape est donc un ensemble disjoint d'index possédant un gain non négatif.

La deuxième étape consiste à réaliser une optimisation globale. Ceci consiste à sélectionner parmi l'ensemble des candidats obtenus à l'étape précédente, ceux qui minimisent une fonction de coût total tous en respectant la contrainte d'espace de stockage requis pour les index. La fonction de coût total est calculée comme la différence entre le coût de traitement des opérations de la charge sans optimisation et l'apport obtenu par l'utilisation de tous les index candidats.

1.5.4.12 Travaux de Finkelstein et al.

DBDSGN est un outil expérimental de conception physique des bases de données relationnelles développé par Finkelstein et al. [80] dans le laboratoire de recherche d'*IBM* à San Jose aux états unis.

Compte tenu d'une charge de requête pour le système R [50], constitué d'un ensemble d'instructions *SQL* et leurs fréquences d'exécution, *DBDSGN* suggère des configurations pour des performances optimales.

Chaque configuration se compose d'un ensemble d'index et un ordre pour chaque table. Les états de la charge de travail sont évalués uniquement pour les configurations qui ont un seul index par table.

DBDSGN utilise les informations fournies par l'optimiseur du système R [50] à la fois pour déterminer quelle colonne pourrait être indexable et obtenir aussi des estimations du coût concernant les états d'exécution des différentes configurations.

Finkelstein et al., [80] considèrent que si les estimations des coûts fournies par l'optimiseur sont les coûts réels d'exécution, l'outil trouvera la solution optimale.

Les principes de *DBDSGN* ont été utilisés dans l'outil *RDT* (relationnel Design Tool), un produit commercial d'*IBM*, qui effectue la conception pour *SQL/DS*, un système relationnel basé sur le système R.

Dans ce contexte, un dialogue entre l'outil et le *DBMS* est établi en utilisant une nouvelle instruction *SQL* appelée *SQL EXPLAIN* pour les estimations des coûts et autres renseignements utiles peuvent être obtenues à partir du système R afin qu'un autre système les utilises.

Elle permet aussi de sauvegarder le plan d'exécution (incluant le chemin d'accès), pour une requête donnée, choisi par l'optimiseur de requête dans des tables dite d'explication. Ces tables peuvent être par la suite accessibles par des requêtes *SQL* ordinaires. Le lecteur intéressé peut consulter les travaux de [79] pour plus d'explications.

1.5.4.13 Travaux de Bellatreche.

L'approche présentée par bellatreche [20], utilise un algorithme glouton pour sélectionner un ensemble d'index de jointure sous une contrainte d'espace.

L'algorithme commence par une solution initiale représentant un seul index (une arête de graphe de jointure étiqueté) ayant un poids maximal. Une fois cet index sélectionné, l'algorithme essaye d'ajouter d'autres nœuds à cet index. L'objectif de cette opération est de réduire le coût d'exécution des requêtes.

Lorsque plus aucune opération d'ajout n'est possible, l'algorithme essaye de réduire la taille de l'index en éliminant certains nœuds. Le but de cette élimination est de réduire le coût d'exécution des requêtes et/ou le coût de stockage de l'index. Les deux opérations sont appliquées afin de réduire le coût d'exécution des requêtes si le coût de stockage de l'index sélectionné ne dépasse pas la contrainte d'espace.

Ensuite, l'algorithme sélectionne, par une démarche itérative, une arête du graphe étiqueté ayant un poids maximum et qui ne représente pas l'index obtenu précédemment. Il applique alors les opérations d'ajout (une ou plusieurs fois) et d'élimination (une ou plusieurs fois) tant qu'il y a une possibilité de réduction dans le coût d'évaluation des requêtes et que la contrainte d'espace est satisfaite. Cette génération ne s'arrête que si les K index sont identifiés (K étant fixé au départ) ou que la capacité de stockage soit violée.

1.5.4.14 Travaux à base d'approches data mining

Toutes les approches proposées pour la sélection d'index à l'aide de techniques data mining préconisent d'effectuer un parcours du treillis de motifs [59], [216], [215] afin de réduire la complexité des algorithmes de sélection des index.

Travaux de Aouiche et al. Les travaux proposés par Aouiche et al, [9], [7] préconisent d'utiliser l'approche d'extraction des itemsets fréquents fermés qui est issue de la théorie des concepts formels [90].

L'étape de sélection des index candidats se base sur l'algorithme *Close* [170], qui permet d'extraire l'ensemble des items fréquents fermés (voir chapitre 2) à partir d'un contexte d'extraction obtenus d'un ensemble de requêtes qu'on considère représentatives.

L'idée de base de cette approche est que plus un attribut ou un groupe d'attributs est fréquemment présent dans la charge de requêtes plus il est intéressant de le(s) considérer dans le processus de sélection des index. Cette fréquence d'apparition, appelé *support* est évaluée par rapport à un seuil minimum noté *minsup* fixé a priori. La sélection finale est exécutée par la suite en considérant les index fournissant le meilleur temps d'exécution.

Pour avoir une vue sur l'approche d'extraction proposés par Aouiche et al, [9], [7], considérons l'exemple ci-dessous.

Exemple. Soit un ensemble de cinq requêtes définies sur un schéma en étoile, composé de deux tables de dimension channels et customers et une table des faits *sales*. Supposons que *minsup*=3 (en valeur absolue).

(1) Select sales.channel_id, sum(sales.quantity_sold) sales.channel_id; from sales, channels where sales.channel_id=channels.channel_id and channels.channel_desc='Internet' group by sales.channel_id ;
(2) select sales.channel_id, sum(sales.quantity_sold), sum(sales.amount_sold) from sales, channels where sales.channel_id=channels.channel_id and channels.channel_desc ='Catalog' group by sales.channel_id ;
(3) select sales.channel_id, sum(sales.quantity_sold),sum(sales.amount_sold) from sales, channels where sales.channel_id=channels.channel_id and channels.channel_desc ='Partners' group by sales.channel_id ;
(4) Select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id=customers.cust_id and customers.cust_gender='M' group by sales.cust_id ;
(5) Select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id=customers.cust_id and customers.cust_gender='F' group by sales.cust_id ;

TABLE 1.4: Exemple de charge de requêtes

Pour faciliter la présentation de notre exemple, nous considérons des abréviations concernant les attributs extraits par l'étape de prétraitement des requêtes.

Abréviations	attributs
A1	sales.channel_id
A2	channels.channel_id
A3	channels.channel_desc
A4	sales.cust_id
A5	customers.cust_id
A6	customers.cust_gender

TABLE 1.5: Liste des abréviations des attributs de notre exemple 2

L'approche de sélection des index candidats en utilisant l'algorithme *Close* [170], retourne les résultats suivants :

1-Item	*support*	Itemsets fréquents fermés
A1	3	A1A2A3
A2	3	A1A2A3
A3	3	A1A2A3
A4	2	A4A5A6
A5	2	A4A5A6
A6	2	A4A5A6

TABLE 1.6: Liste des itemsets fréquents fermés extraits par l'algorithme *Close*

Supposons que *minsup*=3 (en valeur absolue). Un algorithme de sélection des index candidats en utilisant une approche basée sur l'extraction des itemsets fermés fréquents, comme *Close* utilisée dans [7], retourne un index de jointure binaire construit sur la table *Channels* et *Sales* sur l'attribut *channels.channel_ desc* (figurant 3 fois dans prédicats de sélection des requêtes).

L'index de jointure entre la table de dimension *customers* et la table des faits *sales* (sur l'attribut *gender*) est élagué, car il n'est pas fréquent (il ne figure que deux fois).

Travaux de Azfeck et al, Les travaux proposés par Azfeck et al., [12] préconisent d'utiliser une approche dynamique basée sur l'extraction des itemsets maximaux.

L'étape de sélection des index candidat se base sur l'algorithme *GenMax* [97], qui permet d'extraire de façon progressive les itemsets fréquents maximaux à partir d'un ensemble de requêtes qu'on considère représentatives.

L'idée de base de cette approche reste la même à savoir que plus un attribut ou un groupe d'attributs est fréquemment présent dans la charge de requêtes plus il est intéressant de le(s) considérer dans le processus de sélection des index. Le principal avantage de cette approche est qu'elle permet de mettre à jour l'ensemble des index sélectionné lorsque la charge de requêtes évolue plutôt que de la recréer à partir de zéro.

L'algorithme GenMax [97], exploite la charge de requêtes pour produire une base de connaissances P qui stocke, par exemple, la liste des itemsets fréquents maximaux, des itemsets fréquents mais non-maximaux, la liste des requêtes etc.

Cette base de connaissances est par la suite exploitée et mis à jour à chaque période grâce à la liste des nouvelles requêtes utilisée ou celle supprimées. Ainsi, lorsqu'on juge qu'il y a nécessité de mettre à jour la liste des index utilisée, l'extraction s'effectuera en utilisant l'ensemble des connaissances, afin d'éviter de réexécuter l'algorithme du début jusqu'à la fin par rapport à la nouvelle charge de requête. Ce qui peut amener à recréer des index qu'on vient de supprimer et qui sont toujours pertinents ce qui engendre un temps d'exécution supplémentaire. Enfin, la sélection finale est exécutée par la suite en considérant les index fournissant le meilleur temps d'exécution.

1.5.5 Bilan et discussion

A la différence des techniques d'optimisation utilisées dans les systèmes de type *OLTP*, les structures redondantes sont plus adaptées pour améliorer les performances des requêtes de type *OLAP*.

Dans le contexte des entrepôts de données relationnels schématisés en étoile, l'index de jointure binaires présente de meilleures performances pour le traitement d'une requête *OLAP* tout en fournissant une taille minimisé par rapport aux autres structures d'index. L'index de jointure binaire permet d'indexer une ou plusieurs tables sur leurs attributs non clés, c'est pour cette raison que nous l'utilisons pour construire notre structure d'index.

Les travaux existants concernant la sélection d'index sont très nombreux que ce soit dans le contexte des bases de données relationnelles ou dans celui des entrepôts de données. Le tableau ci-dessous résume et classifie ces derniers.

Approche	Heuristique		Data mining
	Manuelle	Automatique	
Travaux	- kartika et al.,(2003) - Gûndem et al.,(1999) - Frank et al.,(1992) - Choenni et al.,(1993) - Wang et al.,(1983)	- Chaudhuri et Narasayya (1997) - Golfarelli et al.,(2002) - Feldman et al.,(2003) - Valentin et al.,(2000)	Aouiche et al.,(2005) Azfack et al.,(2006)
Critère	basés sur l'optimiseur de requête ou sur un modèle de coût mathématique		-Fréquence d'apparition des attributs de dimension dans les requêtes - Anti monotonie et monotonie du *support*

TABLE 1.7: Classification des approches de sélection d'index

Nous avons cependant identifié deux points principaux qui sont susceptibles d'être améliorés dans ces approches.

- La sélection des index candidats repose, pour la plupart des approches, sur l'expertise de l'administrateur de l'entrepôt pour proposer une configuration initiale. Etant données la taille et la complexité des entrepôts de données, une approche automatique est indispensable. C'est d'ailleurs l'option retenue par les travaux les plus récents [7], [12].
- La principale limitation des travaux utilisant une approche data mining est qu'ils considèrent, dans le processus de génération des index candidats, seulement une démarche de sélection centralisée avec la fréquence d'accès des requêtes comme métrique d'élagage. Ce choix peut être discutable, car nous avons fait le lien entre ce problème et le problème de sélection d'un schéma de fragmentation verticale. Les travaux sur la fragmentation verticale ont montré la limite des algorithmes de fragmentation basés sur la fréquence (pour plus de détails voir le travail de Fung et al., [87]).

1.6 Problème de sélection des vues matérialisées

Plusieurs travaux et expériences ont montré que l'utilisation judicieuse des vues matérialisées permet d'améliorer le temps de traitement des requêtes complexes avec des ordres de grandeurs significatifs que ce soit dans le contexte *OLTP* ou *OLAP* [7], [13], [20], [131], [181], [95].

D'une manière générale, il existe trois possibilités pour la sélection des vues à matérialiser [203] :

1. *Matérialiser toutes les vues.* Cette approche reste la meilleure option pour améliorer le temps de réponse des requêtes. Cependant, la matérialisation de toutes les vues, est une option impossible en raison de la contrainte de ressources (espace, temps de maintenance etc.) surtout dans le cas de gros volumes de données.
2. *Ne rien matérialiser.* Cette option est non souhaitable, étant donné les temps de réponse élevés qu'elle engendrera. Ceci est dû au fait qu'on sera obliger d'accéder aux données au niveau le plus bas à chaque fois qu'une requête est posée.
3. *Matérialiser partiellement.* Cette option reste la plus réaliste. Elle consiste à mettre en place des mécanismes à même de sélectionner un ensemble optimal de vues à matérialiser, en considérant leur coût (de stockage, de maintenance etc.).

1.6.1 Formalisation du problème de sélection de vues matérialisées

Le problème de la sélection des vues à matérialiser consiste à déterminer un ensemble optimal de vues à matérialiser permettant l'optimisation du temps de réponse en considérant des contraintes liées à l'espace de stockage des vues ou à leurs coûts de maintenance.

Formellement, le problème de sélection de vue à matérialiser peut être présenté de la façon suivante. Étant donné :

1. un entrepôt de données relationnel,
2. un ensemble de requêtes d'interrogation $Q = \{q_1, q_2, ..., q_m\}$qui constituent la charge de travail (les plus fréquentes) avec leurs fréquences d'accès $f = \{f_1, f_2, ..., f_m\}$.
3. une contrainte de stockage S.

L'objectif est de sélectionner un ensemble de vues à matérialiser $V = \{V_1, V_2, ..., V_l\}$réduisant le coût d'exécution de requêtes et/ou le coût de maintenance des vues sélectionnées sous une certaine contrainte de l'espace de stockage S.

1.6.2 Complexité du problème de sélection de vues matérialisées

Le problème de sélection des vues matérialisées est *NPcomplet* [20], [108] et extrêmement difficile surtout dans le contexte des entrepôts de données où un compromis entre les performances des requêtes et les coûts de stockage et/ou de maintenance des vues doit être pris en compte [214].

Le problème de sélection des vues matérialisées reste toujours ouvert malgré le fait qu'il ait été largement étudié que ce soit dans le contexte *MOLAP* ou *ROLAP* [31], [136], [7],[13], [20],[108],[116], [181]. [106], [138], [181], [228], [199].

1.6.3 Approches pour la sélection des vues matérialisées

Plusieurs approches et algorithmes ont été proposés dans la littérature pour résoudre problème de sélection des vues matérialisées que ce soit dans le cas statique en considérant que l'ensemble des requêtes est connu a priori et qu'il n'évolue pas ou dans le cas dynamique, en considérant que l'ensemble des requêtes est inconnu au départ et que les requêtes sont sujettes à évolutions [7],[13], [20],[108],[116], [181]. [106], [138], [181], [228], [199].

Il existe principalement deux classes pour ce problème de sélection des vues : l'un est le problème de sélection des vues matérialisées sous une contrainte d'espace disque et l'autre est le problème de sélection des vues matérialisées sous la contrainte d'espace disque et du temps de maintenance.

1.6.3.1 Approches de sélection sous une contrainte d'espace disque

Pour de nombreuses applications de la vie réelle, les vues matérialisées, consomment de grandes quantités d'espace disque en raison de la très grande taille des tables constituant ces dernières et qui peuvent atteindre habituellement des centaines de giga-octets ou même des téraoctets). Toutefois, l'espace disque alloué pour stocker les vues matérialisées est toujours limitée.

En pratique, le quota d'espace disque ne permet pas de sélectionner toutes les vues qu'il est possible de matérialiser, ce qui motive le problème de sélection de vues matérialisée sous une contrainte d'espace disque.

Beaucoup de travaux de recherche ont récemment été accompli pour résoudre ce problème; en particulier [116], [106], [107] et [191] ont développé une famille d'algorithmes gloutons pour résoudre ce problème.

Les premiers travaux s'adressant à ce problème sont l'œuvre de Harinarayan et al., [116], qui ont mis au point un algorithme glouton *BPUS* (Benefit Per Unit Space).

A chaque itération, l'algorithme *BPUS* sélectionne la vue apportant le plus grand profit par unité d'espace parmi les autres vues à matérialiser, jusqu'à ce que la contrainte de l'espace disque soit dépassé.

Shukla et al. [191] ont étendu les travaux de [116]. A cet effet, ils ont proposé une sélection plus efficace, selon l'algorithme glouton *PBS* (picking views by size). A chaque itération, l'algorithme *PBS* sélectionne une vue avec la plus petite taille parmi les vues restant à matérialiser, jusqu'à ce que la contrainte d'espace disque soit violée.

Cependant, l'algorithme *PBS* est applicable seulement pour un sous-ensemble particulier du treillis de dépendance qu'ils ont appelé taille restreinte, ou treillis SR-hypercube. Ils ont prouvé qu'un tel treillis fourni la même qualité de solution que celle proposée par [116].

Par la suite, Gupta et al., [107] ont étendu les travaux de Harinarayan [116] à la sélection des vues et des index. Motivé par le fait que, tout comme les vues matérialisées, les index construit sur les vues matérialisées peuvent aussi améliorer efficacement le traitement des requêtes mais exigent des ressources supplémentaires d'espace disque, [107] généralisent ce problème de sélection des vues en problème de sélection des vues et des index. Ils ont

proposé deux algorithmes gloutons pour ce problème et ont prouvé qu'ils fournissent une meilleure solution que celle proposée dans [116].

Gupta [106] a aussi élaboré un cadre théorique pour le problème de sélection des vues en se basant sur l'algorithme glouton proposé dans [116], il a proposé plusieurs heuristiques avec un temps polynomial pour sélectionner des vues avec trois important types de graphes de vue: le graphe *AND*, le graphe *OR* et le graphe générale *AND-OR*, qui sont au-delà du graphe proposé dans [116].

1.6.3.2 Approches de sélection sous une contrainte de temps de maintenance

Un entrepôt de données est toujours sujet à des mises à jour des données. En particulier, la fréquence d'actualisation des données dans certaines applications, peut être considérablement élevée.

Par conséquent, en plus de la contrainte d'espace disque, le temps de maintenance d'une vue est également une contrainte importante pour le problème de sélection des vues dans de nombreuses applications réelles.

A cet effet, le problème de sélection des vues matérialisées sous une contrainte du temps de maintenance a aussi été étudié [108], [181], [57], [143]. En particulier, nous citons l'algorithme évolutionnaire qui a été introduits pour traiter ce problème [214], [140], [229], [234].

1.6.4 Les algorithmes utilisés pour la sélection des vues matérialisées

Selon les travaux de [65] et [195], nous pouvons classer les algorithme utilisés par les approches de sélection des vues matérialisées en quatre grands groupes : (1) Sélection à base d'algorithmes déterministes. Dans ce cas, on construit une solution de façon déterministe pour ensuite appliquer une sorte d'heuristique (par exemple un algorithme glouton) pour réduire l'espace de la solution [116], [108], [109]. (2) Sélection à base d'algorithmes génétiques (3) Sélection à base d'algorithmes aveugles (4) Sélection à base d'algorithmes hybrides.

Dans [107], une extension est proposée, pour améliorer la qualité des résultats en utilisant des index sur les vues sélectionnées. Dans [191], une méthode fondée sur le calcul préalable d'un sous-ensemble d'agrégats qui permet d'avoir un bon résultat, mais pas aussi optimal par rapport aux approches heuristiques. Cependant, ces méthodes ne sont efficaces que lorsque le nombre de vues est relativement faible.

1.6.4.1 Sélection à base Algorithmes génétiques (GA)

Les algorithmes génétiques ont été introduits dans [140], [233], afin d'obtenir de meilleures solutions pour un plus grand nombre de vues respectant les contraintes de maintenance des vues et de traitement des requêtes.

L'idée de base est de commencer avec une population initiale aléatoire et de générer des descendants par des variations aléatoires (par croisement et mutation). Les 'plus fort' membres de la population survivent pour les sélections ultérieures.

L'algorithme prend fin dès que l'on n'a plus d'amélioration par rapport à une période donnée ou après un nombre prédéterminé de générations. Les individuels qui se trouvent plus fort sont la solution.

Toutefois, la possibilité d'une solution infaisable crée certains problèmes. En fait, l'approche proposée dans [233] ne contient pas une méthode pénalisante pour décourager les solutions infaisables. Cette lacune a ensuite été abordée dans [140].

1.6.4.2 Sélection à base d'algorithmes aveugles

Les Algorithmes de cette catégorie suivent une tout autre démarche. Ainsi, une série de mouvements, est définie et ces mouvements constituent les bords entre les différentes solutions. Deux solutions sont reliées par une arête Si (et seulement si) elles peuvent être exactement transformées l'une en l'autre en un seul mouvement.

Le recuit simulé est un exemple type d'algorithme aveugle qui passe au hasard le long des arêtes selon un calendrier de refroidissement et se termine lorsqu'aucune solution applicable n'existe ou bien lorsque toute l'énergie dans le système est perdue (état congelé).

Le recuit simulé a été appliqué au problème de sélection des vues dans [64], [128]. Ainsi, [64] montrent que le coût d'un ensemble de vues matérialisées sélectionné en utilisant un recuit simulé est jusqu'à 70% moins que lorsqu'on utilise des approches à base d'algorithmes génétiques [233] et d'algorithmes heuristiques [233], [228].

1.6.4.3 Sélection à base d'algorithmes hybrides

Les algorithmes hybrides combinent les stratégies purement déterministes et des algorithmes purement aléatoires (aveugle). Les solutions obtenues par les algorithmes déterministes sont utilisés comme points de départ pour les algorithmes aléatoires ou comme population initiale pour les algorithmes génétiques. Dans [108], l'approche hybride a été appliquée pour le problème de sélection des vues.

D'autres approches ont essayé d'allier la puissance des algorithmes génétiques à la recherche heuristique pour trouver le meilleur ensemble de vues matérialisées.

Dans [64], [233], [228], [108] l'algorithme génétique et de recuit simulé sont utilisés pour trouver le meilleur ensemble de vues intermédiaires dans un graphe appelé plan multiple d'exécution des vues ($MVPP$) [228]. Ceci est réalisé dans le but de réduire au minimum le coût de traitement des requêtes et de maintenance des vues. Toutefois, le nombre de vues dans le graphe $MVPP$ est relativement faible et ne peut dépasser les 60 requêtes et 250 vues.

Dans [140], des algorithmes génétiques et heuristiques ont été proposés pour sélectionner le meilleur ensemble de vues à matérialiser à partir d'un graphe AND/OR de vues [108]. Cependant, le nombre de nœuds joint dans le graphe AND/OR ne dépasse pas les 250.

1.6.5 Travaux sur la sélection de vues matérialisées

Plusieurs chercheurs se sont intéressés au problème de sélection des vues matérialisées [116], [228], [134], [13], [153], [7]. Nous présentons, sans toutefois prétendre être complet,

un aperçu de ces travaux en détaillant un peu plus ceux qui utilisent une approche data mining.

1.6.5.1 Travaux de Harinarayan et al.

Harinarayan et al.,[116] présentent le problème de sélection des vues matérialisées sous forme d'un treillis formé par un ensemble de vues V et un opérateur de dépendance capturant les relations existantes entre les vues. A partir du constat que certaines vues peuvent être calculées à partir d'autres vues, les auteurs définissent ceci comme une dépendance entre vues.

La racine du treillis correspond à la vue dont laquelle toutes les vues sont dépendantes (la totalité du cube) et sa borne inférieure correspond à la vue générée en agrégeant complètement les données de n'importe quelle vue du treillis. À chaque vue est associée le nombre de cellules qu'elle contient. Les vues du treillis, quant à elles, sont liées par des relations d'ancêtre et de descendant.

L'algorithme de sélection *BPUS* (Benefit Per Unit Space), initialise l'ensemble des vues à matérialiser à la vue représentant la racine du treillis de vues. A chaque itération, dont le nombre est fixé à k, l'algorithme *BPUS* sélectionne la vue apportant le plus grand profit par unité d'espace parmi les autres vues à matérialiser, jusqu'à ce que la contrainte de l'espace disque soit dépassé.

Algorithm 1.1 Algorithme de sélection des vuess de Harinarayan et al.

ConfigV ←{vue de la borne supérieure du treillis}
Pour i = 1 à k faire
  Sélectionner une vue v ∉ConfigV tel que B(v,ConfigV) soit maximal
  ConfigV ←ConfigV∪ {v}
fin pour
retourner (ConfigV)

Nadeau et al.[153] ont montré que l'algorithme de Harinarayan et al.,[116] n'est pas optimal car, à chaque itération, tous les noeuds non sélectionnés du treillis de vues sont évalués.

1.6.5.2 Travaux de Yang et al

Yang et al.,[228] proposent une approche permettant de minimiser au mieux les coûts d'évaluation de requêtes et de maintenance des vues sélectionnées.

L'algorithme débute par la construction, grâce à un modèle de coût, d'un arbre algébrique optimal pour chaque requête. Par la suite on fusionne en un même graphe orienté, acyclique et étiqueté appelé Plan Multiple d'Exécution des Vues (*PMEV*), les graphes optimaux des requêtes ayant des sous-expressions communes.

Dans un *PMEV* chaque nœud représente une opération d'une requête Q_i à laquelle il est associée. On associe à chaque nœud du *PMEV*, grâce à un modèle de coût, un poids statique représentant le bénéfice si le nœud est matérialisé. Ce poids est utilisé pour ordonner les nœuds et celui ayant la plus grande valeur sera privilégié pour la matérialisation.

Enfin, on calcule leur poids dynamique qui représente la différence entre le bénéfice si la vue est matérialisée et le coût de maintenance de la vue. La vue est alors matérialisée si son poids dynamique est positif. Cependant, cet algorithme ne considère pas la contrainte de stockage.

1.6.5.3 Travaux de kotidis et al

Kotidis et al., [134] proposent le système *DynaMat* qui sélectionne des vues à matérialiser dans le cas dynamique. Le système enregistre les évolutions des requêtes et matérialise, dans chaque cas, le meilleur ensemble de vues pour satisfaire ces requêtes.

Il permet de mettre à jour les vues sélectionnées si la taille de ces dernières dépasse la capacité de l'espace de stockage allouée par l'administrateur suite aux opérations de mise à jour. Il procède alors à certaines éliminations selon des critères de remplacement. Par exemple, les vues les moins utilisées sont éliminées.

L'originalité de ce système tient à l'évolution dynamique de l'ensemble des vues à matérialiser, contrairement à l'approche statique, où la sélection des vues à matérialiser reste statique.

1.6.5.4 Travaux de Baralis et al

Baralis et al., [13] proposent deux heuristiques pour la construction de la configuration de vues à matérialiser. La première consiste à réduire le treillis des vues candidates en ne considérant que les vues associées aux requêtes de la charge.

La deuxième heuristique élague les vues qui ne contribuent pas à l'amélioration des performances. Cette heuristique est basée sur l'estimation de la taille des vues en utilisant la technique d'estimation de Shukla et al., [191]. Selon la taille estimée des vues, il est possible de déterminer à partir de quel niveau d'agrégation il n'est plus pertinent de matérialiser.

La réécriture de requête peut être effectuée au niveau de l'expression *SQL* ou par concaténation d'arbres (plans d'exécution) Au niveau *SQL*, en remplaçant certaines relations virtuelles du *FROM* par leurs sources et en enrichissant les conditions de la clause *WHERE* pour obtenir le résultat de la question initiale.

Notant que la concaténation d'arbre est le mécanisme consistant à remplacer un nœud pendant dans un arbre relationnel par un autre arbre calculant le nœud remplacé.

1.6.5.5 Travaux de Nadeau et al.

Nadeau et al., [153] proposent un algorithme appelé *PGA* (Polynomial Greedy Algorithm). Il débute par la phase de nomination qui consiste à nommer le nœud fils $v1$ qui a la taille minimale à partir de la borne supérieure du treillis.

L'ensemble des vues candidates contient alors $v1$. Les fils de $v1$ sont examinés pour nommer le meilleur fils. Le fils trouvé est ajouté à l'ensemble des vues candidates.

La seconde itération commence par une autre phase de nomination, partant de l'un des fils non encore nommés de la borne supérieure du treillis. Si plusieurs choix de nomination se présentent, un nœud est pris au hasard.

Cette phase de nomination se poursuit jusqu'à atteindre la borne inférieure du treillis. Un chemin de vues candidates est ainsi construit.

Après la construction du chemin des vues candidates, la phase de sélection intervient pour évaluer chaque vue appartenant à un ensemble des vues candidates et sélectionne d'une manière gloutonne celle apportant le bénéfice maximal.

L'alternance entre la phase de nomination et de sélection se poursuit jusqu'à atteindre les contraintes de sélection comme l'espace disque alloué au stockage des vues matérialisées.

1.6.5.6 Travaux de Aouiche et al.

Partant du constat que les charges de requêtes dans le contexte des entrepôts de données, ont tendance à être volumineuses et leur coût de traitement est par conséquent important.

Aouiche et al., [8], [10], [7], au lieu de réaliser l'optimisation directement à partir de la charge, proposent une stratégie de sélection de vues matérialisées basée sur la classification non supervisée des requêtes,

Pour réaliser leur stratégie de sélection, les auteurs supposent la présence d'une charge de requêtes représentatives dont on souhaite optimiser le temps d'exécution.

La première étape consiste à construire à partir de la charge un contexte de classification. Ce contexte est modélisé comme une matrice ayant autant de lignes que de requêtes et autant de colonnes que d'attributs extraits des requêtes de la charge.

Par la suite, ils utilisent l'algorithme de classification Kerouac [127] pour regrouper l'ensemble des requêtes qui sont syntaxiquement similaires. Des mesures de similarité et de dissimilarité entre les requêtes permettent d'évaluer l'homogénéité interne des classes et d'hétérogénéité entre classes. Une classe homogène est alors une classe pour laquelle la syntaxe des requêtes est similaire.

Dans chaque groupe de requêtes obtenues par la classification, on procéde à un processus de fusion afin de construire une configuration de vues candidates.

Etant donné qu'à ce stade, chaque requête différente d'une classe donnée est associée à une vue candidate, les auteurs utilisent par la suite un processus de fusion des requêtes de chaque classe afin de réduire ce nombre.

Enfin, pour satisfaire la contrainte d'espace allouée pour stocker les vues sélectionnées, les auteurs proposent un algorithme glouton qui exploite des modèles de coût pour construire la configuration finale des vues. Les modèles de coût qui évaluent le coût d'accès aux données en utilisant les vues, ainsi que le coût de stockage de ces vues.

1.6.6 Bilan et discussion

Le Tableau 1.7 résume la classification des travaux traitant du problème de sélection des vues matérialisées. Il ressort de ce dernier que la plupart des approches proposées essayent de modéliser les relations entre vues en utilisant diverses structures de données (treillis, graphes, ...).

Approche	Travaux	Algorithme
Data mining	Aouiche et al.,(2005)	Classification
Treillis	Nadeau et al.,(2002)	Glouton
	Shukla et al.,(2000)	Glouton
	Harinarayan et al.,(1996)	Glouton
	Baralis et al.,(1997)	Glouton
	kotidis et al.,(1999)	Glouton
Plan	Zhang et al.,(1997)	Génétique
	Yang et al.,(1997)	Glouton

TABLE 1.8: Classification des approches de sélection des vues matérialisées

Etant donné la complexité des traitements, dans le contexte des entrepôts de données caractérisés par la grande volumétrie des données et la complexité des requêtes d'interrogations, ces dernières présentent des limitations.

L'approche à base de la technique de classification [8], [10], [7] souffre de la centralisation des traitements et de l'adoption d'une classification au niveau des requêtes et non pas de vues. Cette démarche peut générer des groupements de requêtes comportant des agrégations de granularité fine (nombre élevé d'attributs dans la clause Group by) car elles génèrent beaucoup de groupes dans un petit nombre de n-uplets. Par conséquent, l'accès à une vue de grande taille devient plus coûteux que l'accès aux tables de base.

1.7 Problème de distribution de l'espace de stockage entre les vues matérialisées et les index

Étant donné que les index et les vues matérialisées accélèrent le temps d'exécution des requêtes et partagent la même ressource de stockage et que plusieurs travaux ont démontré que ces structures utilisées séparément sont insuffisantes pour accélérer et garantir une bonne performance de toutes les requêtes décisionnelles [20], [187], [138],[138]. Un certain nombre de travaux se sont intéressés au problème de répartition de cet espace entre les index et les vues matérialisées.

1.7.1 Formalisation du problème de distribution de l'espace de stockage entre les vues matérialisées et les index

Formellement, le problème de distribution de l'espace de stockage entre les vues matérialisées et les index, peut être présenté de la façon suivante. Étant donné :

1. un entrepôt de données relationnel,
2. un ensemble de requêtes d'interrogation $Q = \{q_1, q_2, ..., q_m\}$,
3. un ensemble d'index $I = \{I_1, I_2, ..., I_p\}$ déjà sélectionnés,
4. un ensemble de vues $V = \{V_1, V_2, ..., V_l\}$ déjà sélectionnés,
5. une contrainte de stockage S.

L'objectif est de sélectionner un ensemble d'index et de vues réduisant le mieux le coût d'exécution des requêtes et respectant la contrainte d'espace de stockage S.

1.8 Complexité de la distribution de l'espace disque entre les vues et les index

La distribution de l'espace disque entre les vues et les index dans l'objectif d'améliorer les performances des requêtes, est un problème difficile. Dans [20] plusieurs facteurs sont énumérés pour expliquer les raisons de cette difficulté :

- La nécessité d'avoir une métrique pour décider de la distribution d'espace entre les vues et les index.
- L'interdépendance mutuelle entre les deux problèmes.
- La nécessité de tenir compte des répercussions des opérations de mise à jour sur les vues matérialisées et les index [120]. Ainsi, les tailles des vues et des index peuvent augmenter ou diminuer. Il est donc nécessaire de redistribuer l'espace entre les vues et les index afin de garantir une meilleure performance.
- La nécessité de tenir compte de l'évolution des fréquences des requêtes qui peuvent modifier l'intérêt de certaines vues ou de certains index et par conséquent entraîner une redistribution de l'espace.

1.8.1 Approches pour la distribution d'espace de stockage entre les vues matérialisées et les index

Un certain nombre de travaux se sont intéressés au problème de distribution d'espace entre les vues matérialisées et les index.

1.8.1.1 Travaux de Bellatreche et al.,

Bellatreche et al. [20] ont proposés une méthodologie de distribution automatique d'espace entre les vues et les index dans le contexte statique.

Cette méthodologie est basée sur une approche itérative. Elle commence par une sélection initiale des vues et des index, puis essaye d'améliorer cette dernière en utilisant deux algorithmes gloutons (espion des index et espion des vues).

L'approche reconsidère itérativement la solution initiale obtenue dans le but de réduire davantage le coût d'exécution des requêtes en redistribuant l'espace de stockage entre les vues et les index.

Elle s'appuie sur une compétition entre deux agents, l'espion des index et l'espion des vues. L'espion des index (respectivement, des vues) vole de l'espace réservé pour stocker les vues (respectivement, les index). L'espace ainsi récupéré est utilisé pour créer d'autres index à la place des vues élaguées, suivant des politiques de remplacement.

L'opération est validée si le coût d'exécution des requêtes est réduit. La sélection d'index et de vues commence par l'utilisation de l'espion qui réduit le plus le coût des requêtes. Le processus de sélection s'arrête lorsqu'il n'y a plus de réduction du coût des requêtes.

1.8.1.2 Travaux de Rizzi et Saltarelli,

Rizzi et Saltarelli ont proposé une approche qui détermine a priori un compromis entre l'espace de stockage alloué aux index et aux vues matérialisées en se basant sur les requêtes de la charge [181].

L'idée de Rizzi et Saltarelli est que le facteur clé dans l'optimisation des performances des requêtes et leur niveau d'agrégation, défini par la liste des attributs de la clause *Group by* et la sélectivité des attributs présents dans les clauses *Having* et *Where*.

En effet, la matérialisation offre un grand bénéfice aux requêtes comportant des agrégations de granularité grossière (nombre faible d'attributs dans la clause *Group by*) car elles génèrent peu de groupes dans un grand nombre de n-uplets et, par conséquent, l'accès à une petite vue est moins coûteux que l'accès aux tables de base.

D'autre part, les index donnent leur meilleur bénéfice avec des requêtes contenant des attributs dont la sélectivité est élevée, car elles sélectionnent peu de n-uplets et, par conséquent, l'accès à un nombre élevé de n-uplets inutiles est évité.

Les requêtes avec des agrégations fines et de fortes sélectivités encouragent l'indexation. En revanche, les requêtes avec des agrégations grossières et de faibles sélectivités encouragent la matérialisation.

1.8.2 Bilan et discussion

Les vues et les index sont des structures redondantes qui sont souvent utilisées pour l'optimisation du temps de traitement des requêtes d'interrogation des entrepôts de données

Etant donnée les effets d'interaction entre ces deux structures redondantes, une distribution automatisée de l'espace de stockage s'avère primordiale. Parmi les travaux qui se sont intéressés à cette problématique, l'approche Bellatreche et al., [20] identifie les vues ou les index à éliminer en utilisant des politiques basées soit sur la fréquence, ou la taille. Ceci se traduit par quatre démarches différentes :

(1) la vue la moins fréquemment utilisée,

(2) l'index le moins fréquemment utilisé,

(3) la vue de plus petite taille,

(4) l'index de plus grande taille.

L'approche de Rizzi et Saltarelli [181] détermine a priori un compromis entre l'espace de stockage alloué aux index et aux vues matérialisées en se basant sur le niveau d'agrégation des requêtes de la charge.

Ainsi la matérialisation offre un grand bénéfice aux requêtes comportant des agrégations de granularité grossière et de faibles sélectivités et l'indexation est plus adaptée pour les requêtes avec des agrégations fines et de fortes sélectivités.

Néanmoins, nous estimons qu'une politique basée sur des critères de coût (en éliminant les vues ou les index qui contribuent le moins dans la réduction du coût d'évaluation des requêtes) serait plus intéressante puisqu'elle permettrait d'octroyer l'espace sur un critère de mérite.

1.9 Conclusion

Dans ce chapitre nous avons présenté les principaux concepts et travaux relatifs à l'optimisation des entrepôts de données. Tout d'abord, nous avons présenté les caractéristiques d'un entrepôt et décrit ensuite les différents modèles utilisés pour le concevoir.

Nous avons abordé la problématique d'optimisation des entrepôts de données relationnels et les principales techniques permettant d'améliorer les performances d'exécution des requêtes décisionnelles, à savoir: l'indexation, les vues matérialisées et la fragmentation.

Par la suite, nous avons parlé du problème de sélection des structures d'optimisation redondantes (index et vues matérialisées) et donné un descriptif des principaux travaux réalisés dans ce contexte. Nous décrivons et détaillons dans le prochain chapitre, les différentes techniques du data mining qui ont un intérêt pour notre travail.

Chapitre 2

Les approches data mining

2.1 Introduction

Nous présentons dans ce chapitre un tour d'horizon des algorithmes et méthodes de clustering et de classification associative basée sur le treillis d'itemsets. Nous définissons ces deux problèmes et les concepts qui s'y rapportent. Nous décrivons par la suite les plus importantes méthodes existantes pour résoudre ces deux problèmes, en mettant l'accent en particulier sur les méthodes qui sont les plus en relation avec cette thèse. Ainsi pour l'approche de clustering, nous présentons les méthodes de partitionnement (algorithme *Kmeans*). Pour la classification associative basée sur le treillis d'itemsets, nous présentons l'approche d'extraction des itemsets fréquents (algorithme *Appriori*), fréquents fermés (algorithme *Close* et *Charm*) et maximaux (algorithme *GenMax*).

2.2 Définition du data mining

Le data mining, appelé aussi fouille de données ou de manière générique l'Extraction de Connaissance à partir de Données (*ECD*), est un domaine de recherche qui a véritablement pris son essor au milieu des années 90, suite au premier atelier sur le Knowledge Discovery in Data Bases, organisé par Frawley et Piatetsky-Shapiro [86].

Le data mining est défini comme le traitement d'une grande quantité d'informations afin d'y extraire des connaissances non triviales, implicites, potentiellement utiles [114], [72]. [168].

Enfin, Eric Brethenoux qui est le directeur de la recherche pour les technologies avancées au sein du *Gartner Group* le défini comme "un processus de mise à jour de nouvelles corrélations, tendances et de modèles significatifs par un passage au crible des bases de données volumineuses et par l'utilisation de modèles d'identification technique aussi bien statistiques que mathématiques".

Le data mining grâce aux méthodes de structuration [25], d'explication ou de prédiction [235]permet d'extraire plusieurs formes de connaissances à partir des données. Ces connaissances peuvent prendre la forme de règles, de modèles, de régularités, de concepts... etc.

2.3 L'approche de clustering

Dans l'approche de clustering ou classification non supervisée, on dispose d'un ensemble d'objets non classés a priori. L'objectif est alors d'obtenir des groupes (clusters) d'objets homogènes, en favorisant l'hétérogénéité entre ces différents groupes. Ceci consiste à augmenter la cohésion interne d'un même cluster (que les objets appartenant à ce cluster soient les plus similaires possible) et son isolation externe (que les objets appartenant à des clusters distincts soient les plus distincts possible) [114].

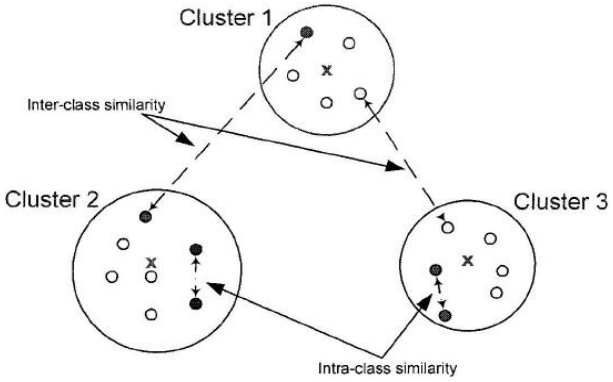

FIGURE 2.1: similarité intra et inter-cluster

2.3.1 La nature combinatoire du clustering

La classification non supervisée d'un ensemble d'objets est un problème hautement complexe. En effet, le nombre de façons de classer n objets en k classes est donné par Liu [145]:

$$P_{n,k} = \frac{1}{k!}\sum_{i=1}^{k} C_k^i(-1)^{k-i}i^n \tag{2.1}$$

A titre illustratif, si nous devions trouver une partition optimale de 15 objets en 7 classes, il faudrait énumérer et évaluer la qualité (au sens d'un critère particulier) d'un peu moins de cinq millions de partitions possibles !

$$P_{15,7} = 4729725$$

Si l'on devait envisager toutes les partitions possibles sans fixation de nombre de classes, il faudrait énumérer un nombre de cas égal au nombre de Bell :

$$B_n = \sum_{i=1}^{n} P_{n,k} = e^{-1}\sum_{i=1}^{\infty}\frac{k^n}{k!} \tag{2.2}$$

Ce nombre revient à envisager toutes les partitions possibles de la plus fine où tous les objets sont isolés, à la plus dense, où tous les objets sont regroupés dans une classe unique.

A titre d'exemple, le nombre de partitions possibles dans une population de 15 objets est de l'ordre de 1,4 milliards.

$$B_{15} = 138295854515$$

Nous voyons que le nombre de partitions possibles sans fixation du nombre de classes est largement supérieur au nombre de partitions possibles avec fixation du nombre de classes. Cependant, fixer un nombre de classes ne simplifie pas le problème de la classification non supervisée pour autant. En effet, le problème reste dans tout les cas, hautement coûteux en temps de calculs.

2.3.2 Les différentes méthodes en clustering

De très nombreuses méthodes de clustering existent et de nouvelles méthodes ou des améliorations de méthodes existantes sont proposées régulièrement par la communauté de la fouille de données. Il est ainsi possible de regrouper ces approches selon des caractéristiques communes. Nous adoptons une taxonomie inspirée des articles d'état de l'art dans le domaine [25], [123], [179],[227].

Les méthodes peuvent être séparées en quatre groupes : la première distinction à faire concerne le type de résultat obtenu. Suivant les méthodes, les clusters obtenus peuvent être des ensembles durs ou flous. Certains objets peuvent ne pas être classés et certains clusters peuvent se recouvrir. De plus, le résultat n'est pas forcément plat et peut se présenter sous la forme d'une hiérarchie.

Les algorithmes de clustering diffèrent également par la stratégie mise en place pour construire les clusters. La notion de similarité est utilisée par une part importante des approches. Cependant, d'autres méthodes à base de densité ou de modèles probabilistes existent.

2.3.2.1 Classification suivant le type de résultat obtenu

Clustering dur Dans un clustering dur (hard clustering), chaque élément appartient à un cluster unique. Formellement, ceci consiste à diviser l'ensemble des données X en un ensemble de K clusters, $C = \{C_1, C_2, .., C_k\}$, formant une partition de X. Ceci correspond à dire que $\cup_{i=1}^{k} C_i = X$ et $C_i \cap C_{i'} = \emptyset \ \forall i \neq i'$.

Ce type de résultat est le plus couramment utilisé, car il est le plus facilement interprétable par l'expert. Cependant, il peut être nécessaire de donner plus de flexibilité aux clusters. En effet, il peut arriver que certains objets se distinguent de manière trop significative des autres objets et leur affecter un cluster peut perturber le processus de clustering. Il arrive que ces objets soient rejetés et qu'aucun cluster ne leur soit affecté dans le résultat final. On parle alors de clustering dur partiel, c'est-à-dire que chaque objet appartient à un ou aucun cluster.

Clustering souple Il peut arriver que la frontière entre les clusters peut être difficile à définir et que certains objets soient à la frontière de plusieurs clusters.

Pour pouvoir refléter ce type d'appartenance, le clustering souple (soft clustering) permet à chaque objet d'appartenir à un ou plusieurs clusters. On peut alors parler de cluste-

ring souple partiel si dans le résultat, un élément peut appartenir à aucun, un ou plusieurs clusters.

Clustering flou L'appartenance à plusieurs clusters est cependant difficile à interpréter pour l'expert. En effet, plus les objets vont appartenir à de nombreux clusters, plus le résultat va perdre en précision et va rendre difficile son interprétation. Le clustering flou apporte alors une solution, en permettant à chaque élément d'appartenir à chacun des clusters selon un certain degré d'appartenance. La somme des valeurs d'appartenance d'un élément aux différents clusters doit être égale à 1.

Il est toujours possible de revenir à un clustering dur en sélectionnant pour chaque objet le cluster dont l'appartenance est maximale.

2.3.2.2 Classification suivant les résultats obtenus

Il existe plusieurs approches pour classifier les méthodes de clustering suivant les résultats obtenus. Nous adopterons la taxinomie proposée dans [123] :

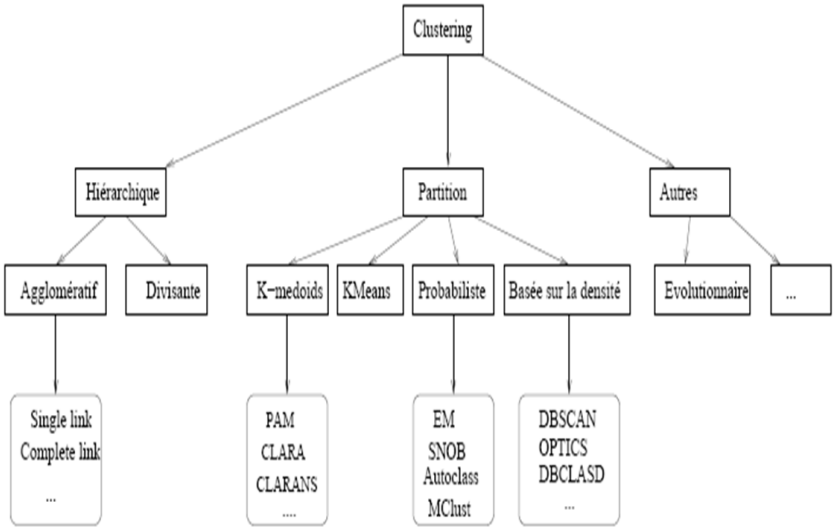

FIGURE 2.2: Taxinomie des algorithmes de clustering

Le clustering hiérarchique Ces méthodes cherchent à former une hiérarchie de clusters. De telle façon que plus on sera bas dans la hiérarchie, plus un cluster contiendra un faible nombre d'objets, mais qui seront plus similaires. L'ensemble des clusters étant généralement représenté par un arbre.

Un objet appartient à une et une seule feuille dans la hiérarchie, mais également à son nœud père et ainsi de suite jusqu'à la racine. Deux grands types d'approches de clustering hiérarchique existent : les approches par agglomération (ou ascendantes) et les approches par division (ou descendantes).

1. *Le clustering hiérarchique divisant.* Les démarches hiérarchiques divisantes ou descendantes démarrent en initialisant le premier cluster à la partition contenant tous les

objets et procèdent par la suite par segmentation de classes jusqu'à ce qu'un critère d'arrêt soit satisfait.

2. *Le clustering hiérarchique agglomératif.* Dans le cas d'une démarche agglomérative ou ascendante, la méthode démarre avec autant de clusters que d'objets puis procède successivement par fusion des clusters jusqu'à ce qu'un critère d'arrêt soit satisfait. Le résultat est alors un arbre appelé dendrogramme dont chaque niveau correspond à une partition particulière du jeu de données. L'avantage de ce type de méthode est qu'elle n'est soumise à aucune initialisation particulière de paramètre(s) ce qui la rend déterministe. Cependant, ce type de méthode impose le calcul de la matrice des distances de tous les points d'observation avec tous les autres et cette masse de calcul est beaucoup trop importante dans le cas de grand volume de données.

Les deux types de clustering hiérarchique peuvent utiliser trois types de méthodes pour classer les objets :

Méthode basé sur la distance minimale

Dans cette approche, la distance entre un objet et une classe est considérée comme la distance entre cet objet et l'objet qui lui est le plus proche de la classe. La distance entre deux classes est la distance minimum entre deux objets appartenant à chacune des classes.

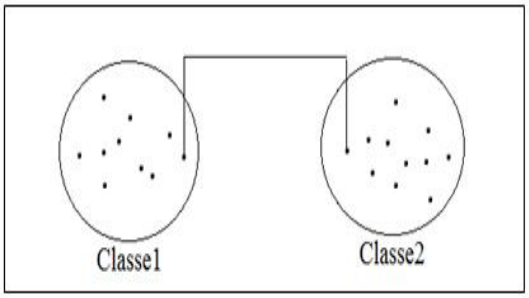

FIGURE 2.3: clustering basé sur la distance minimale

Méthode basée sur la distance maximale

Dans cette approche, la distance entre un objet et une classe est considérée comme la distance entre cet objet et l'objet qui lui est le plus loin de la classe. La distance entre deux classes est la distance maximum entre deux objets appartenant à chacune des classes.

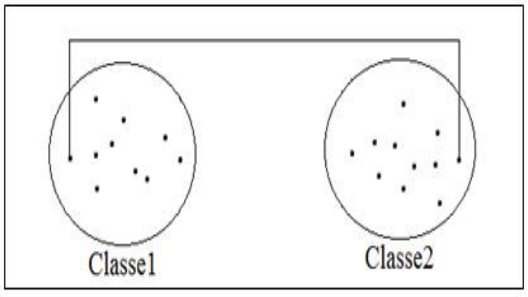

FIGURE 2.4: clustering basé sur la distance maximale

Méthode basée sur la distance moyenne

Dans cette approche, on considère la distance entre deux classes comme étant la distance moyenne qui sépare les deux classes.

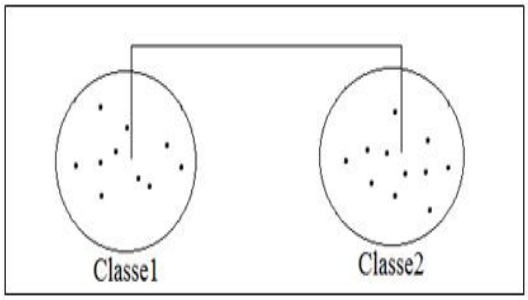

FIGURE 2.5: clustering basé sur la distance moyenne

Le clustering par partition Le but de cette méthode est de trouver la partition de l'espace la plus pertinente pour la formation des clusters. L'une de ces techniques est le clustering par les plus proches voisins, dans laquelle chaque donnée non étiquetée est assignée au cluster de la donnée voisine étiquetée la plus proche, si la distance entre ces deux voisins est inférieure à un certain seuil. Dans le cadre du clustering par partition, plusieurs méthodes existent, on peut citer :

Clustering basé sur la densité. L'idée de cette méthode consiste à former des clusters homogènes constitués de zones de haute densité, entourées des régions de faibles densités. Pour cela, on utilise deux paramètres contrôlant la densité : le radius maximum du voisinage d'un objet (*Eps*) est le nombre minimum de points qui doit être contenu dans ce voisinage (*MinPts*), l'idée ensuite étant d'intégrer à un cluster tout «voisinage dense».

La méthode démarre en sélectionnant aléatoirement l'un des objets de la base. Par la suite et avec une démarche itérative on évalue si le voisinage de l'objet sélectionné respecte le critère de densité, c'est-à-dire s'il y a au moins *MinPts* points dans l'hypersphère de son centre et de rayon *Eps*. Dans le cas positif, on intègre les objets correspondant dans le cluster et répéter le procédé avec ces objets, sinon on sélectionne de façon aléatoire un des objets non encore classés.

Clustering basé sur l'utilisation de grilles L'idée est d'utiliser une grille pour partitionner l'espace en un ensemble de cellules, puis d'identifier les ensembles de cellules denses connectées pour former les clusters. Il existe deux types de ces méthodes :

1. Celles calculant la densité de chaque cellule, puis fusionnant la cellule telle que la résultante soit suffisamment dense (au-dessus d'un certain seuil) et uniforme
2. Celles basées sur la détection des limites des clusters : celles-là détectent d'abord les limites entre zones de haute densité et zones de faible densité, puis reconstituent les clusters à partir de ces limites. Pour ces méthodes, une problématique importante est celle du choix de la taille des cellules :

– des cellules de trop petite taille amènent à une estimation bruitée de la densité (problème du « sur-partitionnement ») ;

– à l'inverse, des cellules de trop grande taille amènent de trop faibles densités (problème du « sous-partitionnement »).

2.3.3 Mesure de distance

Il existe plusieurs mesures permettant de calculer les distances séparant chaque objet de l'autre. Nous détaillerons quelques-unes dans cette partie.

2.3.3.1 Distance de Minkowski

Elle est exprimée par :

$$d(\hat{x}, \hat{y}) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{2.3}$$

Où n représente le nombre d'objets, p représente un nombre entier positif qui permet de donner quelques distances particulières.

2.3.3.2 Distance de Manhattan

La distance de Manhattan est obtenue en affectant la valeur p=1 pour la distance de Minkowski. Ainsi la distance de Manhattan est exprimée par :

$$d(\hat{x}, \hat{y}) = \sum_{i=1}^{n} |x_i - y_i| \tag{2.4}$$

2.3.3.3 Distance euclidienne

La distance euclidienne est obtenue en affectant la valeur p=2 pour la distance de Minkowski. Ainsi la distance euclidienne est exprimée par :

$$d(\hat{x}, \hat{y}) = \sqrt{\sum_{k=1}^{n} (x_i - y_i)^2} \tag{2.5}$$

2.3.3.4 Distance de Hamming

La distance de Hamming [112] entre deux séquences binaires de même taille, est égale au nombre de bits de rang identique par lesquels elles différent.

Ainsi, la distance de Hamming entre deux chaînes de longueur égale est le nombre de positions auxquelles les symboles correspondants sont différents. Autrement, la distance de Hamming est exprimée comme étant la somme des XOR des attributs des objets :

Exemple :

Pour C(1100110, 1010110) a une distance de Hamming =2.

Dans le cas ou C= [000000, 001110,010101,011011,100011,101101]. La distance de Hamming de ce code est définie comme la distance minimale qui sépare deux mots valides du code. Dans ce cas on a une distance de Hamming = 3.

2.3.4 Les algorithmes de clustering

2.3.4.1 Algorithme K-means

K-means [148] est l'algorithme le plus populaire pour les méthodes de clustering dur par partition. Initialement, on identifie K points représentant des clusters (*centroïdes*). Le nombre K est fixé par l'utilisateur. Ensuite, on procède de façon itérative en associant chaque point non classé au cluster dont le *centroïde* est le plus proche. On réévalue par la suite les nouveaux *centroïdes* des clusters jusqu'à ce que les centroïdes ne changent plus significativement.

L'inconvénient de l'algorithme *k-means* est qu'il tend à trouver des classes sphériques de même taille. En présence d'une classe de très petite taille, ou d'une classe prédominante. Cette méthode va donc avoir tendance à vider une classe et la partition ainsi obtenue ne reflétera donc pas correctement la structure des données en classes. Par contre, cet algorithme a l'énorme avantage d'avoir une complexité linéaire par rapport au nombre de données fournies en entrée de l'algorithme. Ceci le rend très est adapté pour des tableaux de grandes tailles.

Algorithm 2.1 Algorithme K-means

1. Placer K points dans l'espace représenté par les objets qu'on veut classifier.
2. Affecter chaque objet au groupe dont le centre de gravité est le plus proche en se basant sur la distance de Minkowski.
3. Recalculer la position de chaque centre de gravité pour les K points
4. Refaire les étapes 2 et 3 jusqu'à ce que les centres de gravités ne puissent plus changer.

2.3.4.2 Algorithme Fuzzy C-means

Cet algorithme développé par [69] et amélioré par [27], appartient à la classe de classification floue. Il permet grâce au coefficient flou d'aider à sortir d'un optimum local. Cependant, son temps de calcul est plus long que celui du *K-Means* et le nombre optimal de classes doit être fixé à l'avance comme pour le *K-Means*.

2.3.4.3 Algorithme CURE (Clustering Using REpresentatives)

CURE [104] est un algorithme de clustering hiérarchique. Dans cet algorithme, chaque classe est représentée par un nombre fixe d'objets. *CURE* utilise une méthode qui combine entre l'échantillonnage (appelé aussi prototypage) et le partitionnement aléatoire. Un point important de CURE est qu'il représente un cluster par un nombre fixe de points dispersés autour de lui. La distance (entre deux clusters) utilisée dans le processus agglomératif est égale au minimum de la distance entre deux représentants dispersés. Le principal problème qui se pose dans cet algorithme est la détermination de la taille de l'échantillon.

2.3.4.4 Algorithme EM (Expectation Maximisation)

EM [34] est un algorithme de classification non supervisée probabiliste. Il consiste à chercher les paramètres d'une loi de mélange comme par exemple le mélange gaussien (ang.

Gaussian mixture) et à maximiser la log-vraisemblance. Comme son nom l'indique, EM se décompose en deux phases : l'expectation et la maximisation.

La phase d'expectation consiste à évaluer la probabilité pour chaque objet en fonction des paramètres du modèle, ainsi que la probabilité d'appartenance de chaque objet à chaque classe. La phase de maximisation consiste à estimer, en fonction des probabilités d'appartenance des objets aux classes, les paramètres qui maximisent la log-vraisemblance des objets.

2.3.5 Caractéristiques et complexité des algorithmes de clustering

La figure ci dessus issue de [6], compare un ensemble d'algorithmes de clustering suivant différentes caractéristiques. Cette table indique les paramètres d'entrées requit par chaque algorithme (2iem colonne), le type de l'ensemble de données s'il est optimisé (3ièm Colonne), la structure du cluster (4ièm Colonne), si elle gère le bruit ou non (5 ième Colonne) et sa complexité de calcul (6 ième Colonne). En général, la Complexité peut être donnée en termes de nombre d'opérations dans la mémoire principale ou le coût d'entrées/sorties nécessaires.

Algorithm	*Input Parameters*	*Optimized For*	*Cluster Structure*	*Outlier Handling*	*Computational Complexity*
k – means	Number of Clusters	Separated Clusters	Spherical	No	$\mathcal{O}(Ikn)$
PAM	Number of Clusters	Separated Clusters, Small Data Sets	Spherical	No	$\mathcal{O}(Ik(n-k)^2)$
CLARA	Number of Clusters	Relatively Large Data Sets	Spherical	No	$\mathcal{O}(ks^2+k(n-k))$
CLARANS	Number of Clusters, Maximum Number of Neighbors	Spatial Data Sets, Better Quality of Clusters than *PAM* and *CLARA*	Spherical	No	$\mathcal{O}(kn^2)$
Hierarchical Methods					
BIRCH	Branching Factor, Diameter Threshold	Large Data Sets	Spherical	Yes	$\mathcal{O}(n)$
CURE	Number of Clusters, Number of Cluster Representatives	Arbitrary Shapes of Clusters, Relatively Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n^2 \log n)$
Density-Based Methods					
DBSCAN	Radius of Clusters, Minimum Number of Points in Clusters	Arbitrary Shapes of Clusters, Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n \log n)$
DENCLUE	Radius of Clusters, Minimum Number of objects	Arbitrary Shapes of Clusters, Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n \log n)$
OPTICS	Radius of Clusters (min,max), Minimum Number of objects	Arbitrary Shapes of Clusters, Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n \log n)$
Miscellaneous Methods					
STING	Number of cells in lowest level, Number of objects in cell	Large Spatial Data Sets	Vertical and Horizontal Boundaries	Yes	$\mathcal{O}(n)$
WaveCluster	Number of Cells for each Dimension, Wavelet, Number of application of Transform	Arbitrary Shapes of Clusters, Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n)$
CLIQUE	Size of the Grid , Minimum Number of Points within each Cell	High Dimensional Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n)$
ScalableEM	Initial Gaussian Parameters, Convergence Limit	Large Data Sets with Approximately Uniform Distribution	Spherical	No (?)	$\mathcal{O}(n)$

FIGURE 2.6: Propriétés des différents algorithmes de clustering

Où n représente le nombre d'objets, k le nombre de clusters, s la taille de simple et l le nombre d'itérations.

2.4 Règles d'association basées sur le treillis de concepts

Dans le cadre de l'extraction des relations entre attributs contenues dans une base de données transactionnelles, l'une des méthodes les plus utilisées consiste à énumérer tous les ensembles d'attributs appelés itemsets, puis à partir de ces derniers, calculer toutes les règles d'association possibles [4], [193], [133], [188],[185].

Ces catégories de méthodes ont été considérés par certains auteurs [47], [48], [113], [142], comme faisant partie des approches de clustering parce que les concepts ne sont pas prédéterminés et les instances utilisées pour l'apprentissage ne sont pas pré-classifiées. Carpineto et Romano [48] ont démontré l'utilité du treillis pour la découverte et la prédiction de classes à partir de plusieurs corpus de données.

2.4.1 Principe général et notion de base

Ce paragraphe définit les principaux concepts liés à la tâche d'extraction des itemsets fréquents. Notre but se limite seulement à expliciter certains concepts qui seront utilisés lors de la présentation de notre stratégie de sélection automatique d'index.

Définition 1 (Contexte d'extraction). Un contexte d'extraction (ou contexte formel) est un triplet *K=(O,I,R)*, décrivant deux ensembles finis O et I et une relation (d'incidence) binaire R entre O et I telle que $R \subseteq O * I$.

L'ensemble O est habituellement appelé ensemble d'objets (ou transactions) et I est appelé ensemble d'attributs (ou items). Chaque couple (o, i) $(o, i) \in R$ désigne que l'objet $o \in O$ possède l'attribut $i \in I$.

Définition 2 (Treillis). Un ensemble ordonné $(T, \leq)$est un treillis si toutes les paires des éléments de T possèdent une borne inférieure et une borne supérieure.

L'ensemble $P(I)$ des parties d'un ensemble I muni de l'inclusion $\subseteq$ est un treillis. Un treillis est complet si tous les sous-ensembles $P \subseteq S$ admettent une borne supérieure et une borne inférieure. Ainsi tous les itemsets des transactions d'une base, peuvent être représentés par un treillis.

Exemple. Treillis des itemsets$\{A, B, C, D\}$.

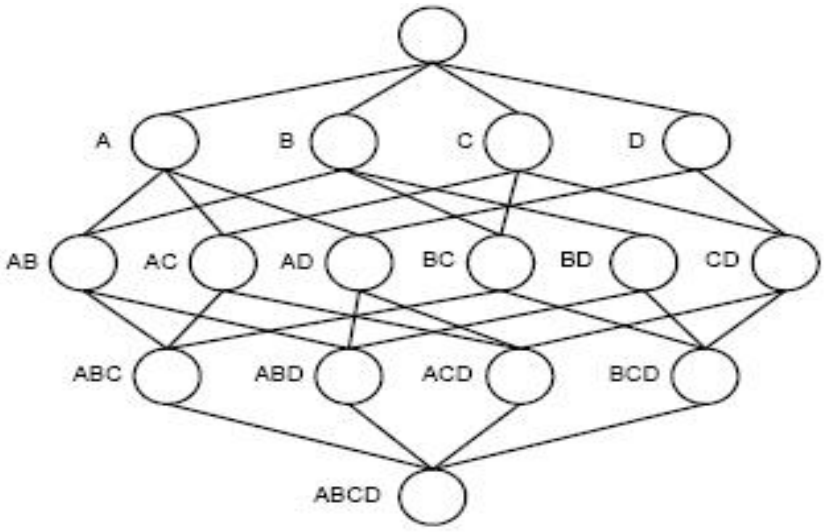

FIGURE 2.7: Treillis d'itemsets

On désignera par la suite la borne inférieure de la paire (x, y) par $(x \vee y)$ et la borne supérieure par $(x \wedge y)$. Les opérations binaires $\vee$ et $\wedge$ sont respectivement $\cup$ et $\cap$.

Définition 3 (Connexion de Galois). Soient deux applications t et i où t associe a un itemset X l'ensemble des transactions contenant X. L'application i associe a un tidset Y l'ensemble des items appartenant a toutes les transactions de Y. Une connexion de Galois est un couple d'applications (i, t) entre l'ensemble des tidsets et l'ensemble des itemsets.

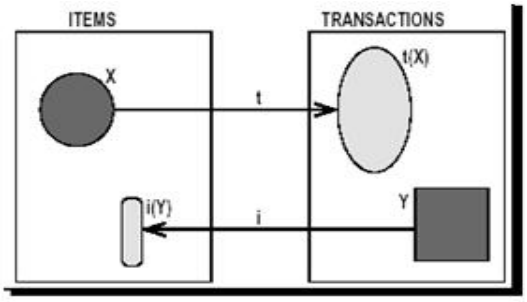

FIGURE 2.8: Connexion de Galois

Définition 4 (Item ou motif). Un item est tout objet, article, attribut, appartenant à un ensemble fini d'éléments distincts $I = i_1, i_2, ..., i_m$. Par exemple, les articles en vente dans un magasin sont des items.

Définition 5 (Itemset). On appelle itemset tout sous-ensemble d'items d'un ensemble fini d'éléments distincts $I = i_1, i_2, ..., i_m$. Un itemset I' de taille K est appelé un *k-itemset*.

Définition 6 (Support d'un itemset). Le support d'un itemset, noté $Sup(X)$ est le nombre de transactions du contexte contenant cet itemset. Le support peut être donné en pourcentage et sa valeur est comprise dans l'intervalle $[0, 1]$.

Définition 7 (Itemset fréquent). Un itemset X est dit fréquent si son support est supérieur ou égal à un seuil minimum noté *minsup* fixé a priori et appartenant à l'intervalle $[0, 1]$. On note : $Sup(X) \geq minsup$.

Définition 8 (Itemset fermé fréquent). La fermeture d'un itemset Y (noté $FCI(Y)$ est le plus grand sur ensemble de Y qui a le même *support* que Y. On note : $Sup(Y) \geq FCI(Y)$. Un itemset fermé i tel que $Sup(i) \geq minsup$ est appelé itemset fréquent fermé.

Définition 9 (Itemset générateur). Le générateur Z d'un itemset fermé X est un sous-ensemble minimal de X tel que sa fermeture est égale à X. Ainsi $ferm(Z) = X$. Ces itemsets sont utilisés afin de construire un ensemble d'itemsets fermés candidats (itemsets fermés potentiellement fréquents) qui sont les fermetures des générateurs [171].

Définition 10 (Bordure positive). On appelle bordure positive B^+ l'ensemble des itemsets fréquents dont tous les sur-ensembles sont peu fréquents. Cet ensemble correspond aux itemsets fréquents maximaux.

Définition 11 (Bordure négative). On appelle bordure négative B^- l'ensemble des itemsets peu fréquents dont tous les sous-ensembles sont fréquents. Ces itemsets correspondent aux itemsets peu fréquents minimaux.

Définition 12 (TidSet). On appelle *tidset* tout sous-ensemble d'identificateurs de transactions (*tids*) de T.

2.4.2 Propriétés de monotonie *et* d'anti monotonie la contrainte des itemsets

Deux propriétés mathématique permettent la réduction significative du nombre d'itemsets engendrés en évitant le parcours inutiles de la base de données lors du calcul du *support*

des itemsets qui sont surement peu fréquents tous en assurant la bonne qualité des itemsets extraits [232].

2.4.2.1 Propriété de monotonie du support

Tout itemset inclus dans un itemset fréquent est fréquent. Ainsi, si un itemset de taille k est fréquent alors tous les sous ensembles de taille k-1 engendrer à partir de ce dernier sont fréquent.

Ainsi, si on considère un itemset fréquent X. C'est à dire que $Sup(X) \geq minsup$ et qu'il existe et $X' \subset X$ alors obligatoirement on aura $Sup(X') \geq minsup$.

2.4.2.2 Propriété de l'anti-monotonie du support

Tout sur-itemset d'un itemset infréquent est infréquent. Ainsi, si un itemset de taille k n'est pas fréquent alors tous les ensembles de taille $k+1$ pouvant être obtenus à partir de ce dernier ne le sont pas. Il n'est donc pas nécessaire de les engendrer.

Ainsi, si on considère un itemset infréquent X. C'est-à-dire que $Sup(X) < minsup$ et qu'il existe $X' \subset X$ alors obligatoirement on aura $Sup(X') < minsup$.

2.4.3 Approches d'extraction basées sur les itemsets fréquents

Toutes les approches proposées pour l'extraction des itemsets fréquents effectuent un parcours du treillis d'itemsets [59], [215], [216] dans le processus de réduction de l'espace de recherche.

[24], [118] les ont classés, selon la stratégie adoptée pour le parcours du treillis et selon la façon de calculer le support des itemsets dans la base de données. Les plus connues sont le parcours en largeur d'abord (Breath First Search) et le parcours en profondeur d'abord (Depth First Search).

Dans le parcours en largeur, le treillis des itemsets est parcouru niveau par niveau, les (k-1)-itemsets sont connus avant de générer les k-itemsets. Les itemsets sont dénombrés selon leurs longueurs d'abord et par ordre lexicographique ensuite.

Le parcours en profondeur d'abord, explore le treillis en profondeur en suivant un arbre structure comme celui de Rymon [184]. A chaque échec, on retourne au nœud précédent l'impasse et on choisit un nouveau nœud.

On retrouve dans la littérature, différents travaux [188], [168], [37], [91], [202] permettant de générer tous les itemsets fréquents dans une base transactionnelle. L'algorithme de référence basé sur cette approche est *Apriori* [4].

Les algorithmes d'extraction qui ont vu le jour par la suite se sont focalisés, pour la plupart d'entre eux, sur la réduction des entrées/sorties et la minimisation du coût de l'étape de calcul du *support* dans le but d'améliorer l'efficacité de la méthode initiale [118]. Les algorithmes les plus connus sont *AprioriTid* [4], Partition [188] et Sampling [202]. Le lecteur intéressé par une description plus complète des méthodes et algorithmes d'extraction des motifs fréquents peut se référer à [171], [15] et [185].

2.4.3.1 L'Algorithme Apriori

L'algorithme *Apriori* proposé par Agrawal et al., [4], permet d'extraire tous les itemsets ayant respectivement un *support* supérieur au seuil minimum, appelé *minsup*. Dans l'algorithme *Apriori* le parcours de la base de données se fait en deux étapes.

Dans la première étape, les *1-itemsets* fréquents sont déterminés en comptant simplement les occurrences des *Items*. La deuxième étape permet d'extraire les itemsets fréquents par un processus itératif qui ne s'arrête que s'il n'y a plus d'itemsets fréquents. Ainsi les itemsets fréquents de l'étape *K*, sont obtenus en supprimant ceux de l'étape *K-1* dont le *support* est inferieur à *minsup*.

Algorithm 2.2 Algorithme Apriori

Entrée : D : Contexte d'extraction,
minsup : support minimum
Sortie : Ensemble des itemsets fermés fréquents
Début
L_1=$\{1 - itemset\, fréquent\}$
Pour $\{k = 2;\ L_{k-1} \neq \emptyset\ ; k++\}$
C_k=*ensemble des nouveaux candidats*
Pour l'ensemble des transaction $t \in D$
Pour tout $k - subset\ s \in D$
Si $(s \in C_k)$ *alors* $count++$;
L_k=$\{s \in C_k \mid s.count \geq minsup\}$;
ensemble de tous les itemsets fréquents=$\cup_k L_k$;

Il faut noter que l'algorithme *Apriori* permet la réduction significative du nombre d'itemsets engendrés en utilisant la propriété de monotonie et de l'anti-monotonie du *support*.

2.4.4 Limites des approches d'extraction des itemsets fréquents

Les algorithmes d'extraction des itemsets fréquents permettent d'élaguer un sous-ensemble du treillis en utilisant le seuil minimum d'élagage (*minsup*) et la propriété d'anti monotonie du *support*.

Cependant, plusieurs études [171], [230], [231], [196], ont montrées que l'approche d'extraction des itemsets fréquents est inapplicable aux données réelles. Ceci est dû au fait que le nombre d'itemsets fréquents généré est exponentiel par rapport au nombre d'attributs considérés.

Ainsi pour une base de N transactions avec t items apparaissant dans cette transaction. Chaque transaction peut donc former C_t^i sous ensembles de taille i.

Etant donné que les approches d'extraction des itemsets fréquents vérifient à chaque itération, si tout sous-ensemble de toute transaction appartient à la liste des candidats.

Ainsi, à l'itération $k-1$, on cherche tous les ensembles de taille k de chaque transaction. Il faut donc, pour chaque transaction, générer les C_t^i sous-ensemble de celle-ci afin de vérifier

leur présence dans la liste des candidats.

On effectuera donc en tout, lors du déroulement d'une telle approche $N \times \sum_{k=2}^{max(t)} \sum C_t^i$ vérification dans le cas où les 2^t itemsets sont fréquents [88]. Ceci montre que le temps de recherche des itemsets est exponentiel vis-à-vis de la taille des transactions.

Ainsi, pour une base de données contenant n attributs booléen le nombre total d'itemsets est égal à $2 * n$. Par exemple, pour un magasin avec 1000 d'articles mis en rayon. Pour un support minimum $minsup = 1$, le nombre d'itemsets pouvant être obtenu est proche de 10^{300}.

Afin de pallier ces insuffisances, la représentation condensé des itemsets a été proposée comme alternatives, pour réduire la taille des itemsets extraits tout en gardant la qualité des résultats obtenus. Deux approches d'extraction condensées des itemsets ont vu le jour : l'extraction des itemsets fermés fréquents et l'extraction des itemsets fréquents maximaux.

2.4.5 Approches d'extraction basées sur les itemsets fermés fréquents

Cette nouvelle approche issue de la théorie des concepts formels [90], a été introduite par N. Pasquier et al.[172].

Cette représentation a connu beaucoup de succès au sein de la communauté scientifique, car elle permet de produire un ensemble d'itemsets fréquents compact d'un point de vue taille et complet d'un point de vue connaissance [15], [23], [94], [196]. Ceci permet d'améliorer les performances et augmente l'efficacité du processus d'extraction grâce à la réduction de l'espace de recherche et la minimisation du nombre de parcours de la base.

Les algorithmes, d'extraction des itemsets fréquents fermés (*FCIs*), peuvent être soit classés selon le fait qu'ils parcourent l'espace de recherche par niveau, ou bien qu'ils essaient de diviser le contexte d'extraction en sous-contextes auxquelles est appliqué le processus récursif de découverte des itemsets fermés.

Les Algorithmes les plus connus, utilisant cette approche sont : *Close* [172], [171], *Closet* [174], *A-Close* [172], *Charm* [231], *Pascal* [16].

2.4.5.1 Algorithme *Close*

L'algorithme *Close* [171] est un algorithme itératif d'extraction des itemsets fermés fréquents qui parcourt l'ensemble des générateurs des itemsets fermés fréquents par niveaux. Close utilise une technique itérative, dite par niveaux, dans la prise en compte des motifs à traiter en s'appuyant sur la propriété d'élagage de l'algorithme *Apriori* et qui stipule qu'un motif fermé fréquent ne peut contenir que des sous-motifs fréquents.

L'algorithme commence par initialiser l'ensemble des 1-générateurs avec la liste des 1-temsets du contexte. Ensuite, l'algorithme exécute un ensemble d'itérations. Durant chaque itération k, la fermeture de tous les k-générateurs ainsi que leurs *supports* sont calculés.

La détermination des fermetures des générateurs, est basée sur la propriété que la fermeture d'un itemset. Celle-ci est égale à l'intersection de tous les objets du contexte le contenant et dont le décompte fournit le *support* du générateur qui est identique au *support* de sa fermeture.

Tous les k-générateurs fréquents, dont le *support* est supérieur ou égal à *minsup*, ainsi que leurs fermetures, sont sauvegardés, les autres (k-motifs non fréquents et les k-motifs non générateurs) sont supprimés. Ainsi, l'ensemble des (*k+1*)-générateurs candidats de l'itération suivante, est construit en joignant les k-générateurs fréquents de l'ensemble des itemsets fermés fréquents identifiés durant l'itération *k*. Les itérations cessent lorsqu'aucun nouveau générateur ne peut être créé et l'algorithme s'arrête alors.

Algorithm 2.3 Algorithme Close

Entrée : *k* : Contexte d'extraction, *minsup*

Sortie : Ensemble des itemsets fermés fréquents

1) Initialiser l'ensemble des candidats de taille 1

2) *Tant que* ensemble des candidats est non vide *faire*

3) *Etape d'élagage*

   1. Calculer le support des candidats
   2. Elaguer l'ensemble de candidats par rapport à *minsup*
   3. Calculer les fermetures des candidats retenus

4) *Etape de construction*

   1. Construire l'ensemble de candidats à utiliser lors de l'itération suivante
   2. Elaguer cet ensemble en utilisant les propriétés structurelles des itemsets fermés

5) *Fin tant que*

6) *Retourner* ensemble des itemsets fermés fréquents

Pour mieux expliciter la démarche d'exploration de l'algorithme *Close,* prenant l'exemple suivant.

Exemple. Soit la base de transaction suivante :

Tid	Item
1	ACD
2	BCE
3	ABCE
4	BE
5	ABCE

TABLE 2.1: Base de transaction

Le treillis des itemsets infréquents et des fermés fréquents est donné par la figure suivante.

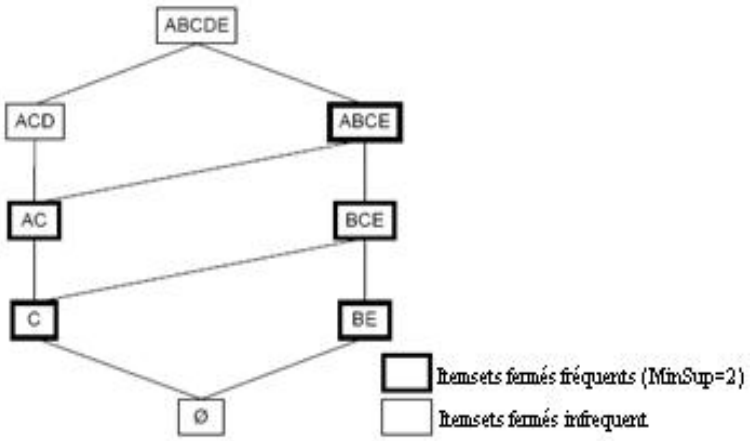

FIGURE 2.9: Treillis des itemsets fermés fréquents

La figure suivante explicite les différentes étapes d'extraction des itemsets fréquents fermés par l'algorithme *Close* pour la même base de transaction donnée par le tableau 2.10 avec $minsup$=2/5.

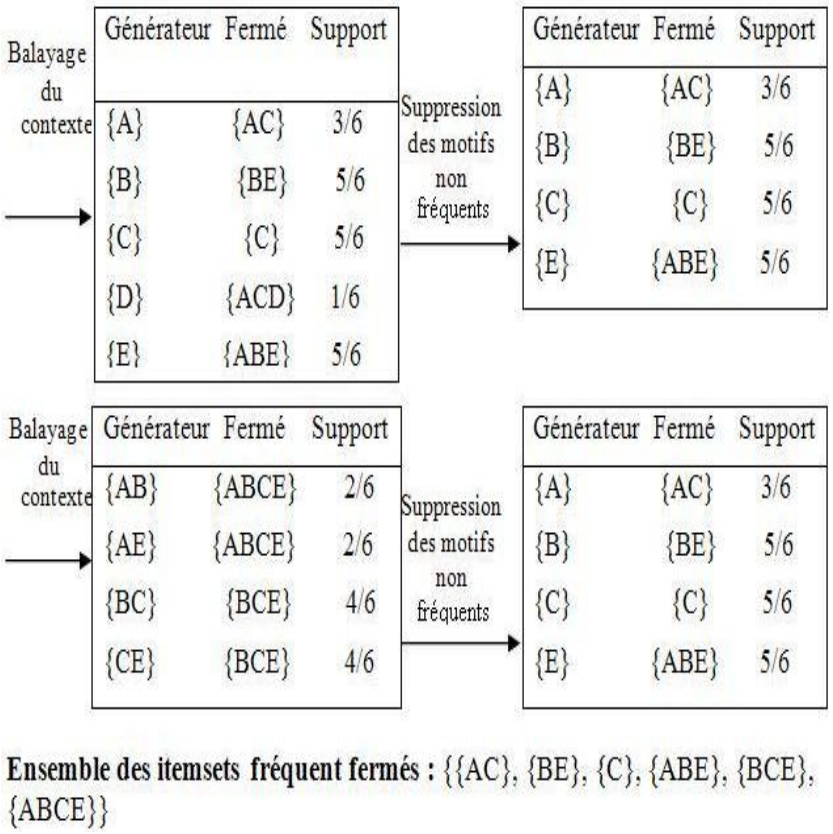

FIGURE 2.10: Etapes d'extraction des itemsets fermés fréquents par l'algorithme *Close*

2.4.5.2 Algorithme *charm*

L'algorithme *Charm* [231] permet l'extraction des itemsets fermés en effectuant une exploration en profondeur d'abord (Depth-first). La spécificité de cet algorithme réside

dans le fait qu'il privilégie une exploration en profondeur d'abord de l'espace de recherche.

L'idée clef est d'exploiter la maximalité d'un itemset fermés, c'est-à-dire un itemset fermé couplé avec l'ensemble des objets le vérifiant n'est pas inclus dans aucun autre itemset fermé.

Contrairement aux algorithmes précédents qui n'explorent que l'espace des itemsets, *Charm* explore simultanément l'espace des itemsets et celui des transactions grâce à une structure appelée *IT-Tree* (ItemSet-TidSet tree). Cette exploration simultanée permet d'identifier rapidement les itemsets fréquents fermés (*FCIs*) et évite d'énumérer plusieurs sous-ensembles non-fermés. Une autre particularité à mentionner au crédit de *Charm*, est qu'il utilise une représentation verticale, appelée *diffset*, pour accélérer le calcul des supports. L'opération fondamentale utilisée est l'union des deux itemsets et l'intersection de leurs *TidSets*.

L'idée de *Charm* est de calculer chaque nœud du treillis pour tester si ses fils sont fréquents ou pas. Les nœuds fils sont calculés en combinant le nœud parent avec tous ses frères.

L'algorithme *Charm* commence par initialiser l'ensemble des nœuds à examiner par les 1-itemsets fréquents et leurs *TidSets*. L'algorithme utilise deux procédures pour explorer et réduire l'espace d'extraction.

Étape d'élagage. *Charm* utilise, dans cette étape, deux principes d'élagage : le premier est basé sur l'élagage des candidats ayant des sous-ensembles non-fréquents (propriété d'anti monotonie de l'algorithme *Apriori*) et le second basé sur l'élagage des candidats non-fermés.

L'étape est implémentée via la procédure *CHARM-PROPRIÉTÉ*. Cette procédure qui se base sur les quatre propriétés décrites ci-dessous, peut modifier la classe courante [P] en supprimant des paires-IT ou en insérant de nouvelles dans $[P_i]$.

Une paire-IT est d'abord élaguée comparativement à *minsup*. Ensuite, on vérifie si elle est maximale ou non. Pour ce faire, il suffit de vérifier que son *Tidset* est inclus dans celui de la paire l'ayant généré. Une fois toutes les IT-paires traitées, la nouvelle classe [Pi] est récursivement explorée en profondeur d'abord, en appelant la procédure *CHARM-ÉTEND*. Cette fonction peut modifier l'ensemble des nœuds courant en supprimant les paires contenues dans d'autres paires.

De nouveaux nœuds fils peuvent être aussi générés et insérés dans l'ensemble des nouveaux nœuds. Si ce dernier n'est pas vide, un appel récursif à CHARM-EXTEND est effectué afin de réaliser un parcours en profondeur d'abord. Toutes les extensions possibles d'itemsets fermés fréquents X d'un itemset Si X_i sont à la fin insérés dans C qui représente l'ensemble des itemsets fermés fréquents. Ce processus est réitéré pour chaque combinaison possible.

La procédure *CHARM-ÉTEND* est responsable de l'exploration de tous les nœuds et de la construction des fils en combinant le nœud courant avec tous ses frères successeurs dans un ordre lexicographique.

Étape de construction. Cette étape est implémentée via la procédure *CHARM-ÉTEND*. Cette dernière est responsable de l'exploration de tous les nœuds et la construction des fils en combinant le nœud courant avec tous ses frères successeurs dans un ordre lexicographique. Cette combinaison s'effectue en faisant l'union des itemsets et l'intersection des TidSets. Le but étant d'élaguer les nœuds n'ayant pas le support minimum tout en appliquant les propriétés citées ci-dessous

- *Propriété 1 :* Si $t(X1) = t(X2)$ alors $t(X1 \cup X2) = t(X1) \cap t(X2) = t(X1) = t(X2)$. Dans ce cas, on remplace toutes les occurrences de *X1* par $X1 \cup X2$ et on enlève *X2* de toutes les considérations ultérieures. En effet, sa fermeture est la même que la fermeture de $X1 \cup X2$.
- *Propriété 2 :* Si $t(X1) \subset t(X2)$ alors $t(X1 \cup X2) = t(X1) \cap t(X2) = t(X1) \neq t(X2)$. Dans ce cas, on remplace toutes les occurrences de *X1* par $X1 \cup X2$ mais on ne peut pas enlever *X2* de toutes considérations ultérieures parce que $t(X1) \neq t(X2)$.
- *Propriété 3 :* Si $t(X1) \supset t(X2)$ alors $t(X1 \cup X2) = t(X1) \cap t(X2) = t(X2) \neq t(X1)$. Dans ce cas, on remplace toutes les occurrences de *X2* par $X1 \cup X2$ mais on ne peut pas enlever *X1* de toutes considérations ultérieures parce que $t(X1) \neq t(X2)$.
- *Propriété 4 :* Si $t(X1) \neq t(X2)$ alors $t(X1 \cup X2) = t(X1) \cap t(X2) \neq t(X1) \neq t(X2)$. Dans ce cas, aucun des deux itemsets ne peut être supprimé, car les deux génèrent des fermetures différentes. Par contre, on ajoute le nœud $X1 \cup X2$ avec son *tidset* associé qui est $t(X1 \cup X2) = t(X1) \cap t(X2)$.

Algorithm 2.4 Algorithme Charm

Entrée : K : Contexte d'extraction, $minsup$

Sortie : FC : Ensemble des itemsets fermés fréquents

1 : $[P] = \{X_i \times (X_i)' : X_i \in \Gamma \wedge \ support(X_i) \geq minsup\}$

2 : CHARM-ÉTEND($[P], FC \neq \emptyset$)

3 : retourner FC

Algorithm 2.5 Procedure CHARM-ETEND

Entrée :$[P]$, FC
Sortie : FC
1 : pour tout $X_i \times (X_i)' \in [P]$ faire
2 : $([P_i] = \emptyset)$ et $(X = X_i)$
3 : Pour tout $X_j \times (X_j)' \in [P] \wedge X_j \geq X_i$ faire
4 : $X = X \cup X_i$
5 : $Y = (X_i)' \cap (X_j)'$
6 : CHARM-PROPRIÉTÉ$([P], [P_i])$
7 : si $[P_i] = \emptyset$ alors
8 : CHARM-ÉTEND$([P_i], FC)$
9 : SUPPRIMER$([P_i])$
10 : $FC = FC \cup X$
11 : fin si
12 : fin pour
13 : fin pour
14 : retourner FC

Algorithm 2.6 Procedure CHARM-PROPRIETE

Entrée : $[P], [P_i]$
Sortie : $[P]$
1 : si $support(X_i) \geq minsup$ alors
2 : si $(X_i)'=(X_j)'$ alors
3 : Supprimer X_j de $[P]$
4 : Remplacer tout X_i par X
5 : sinon
6 : si $(X_i)' \subset (X_j)'$ alors
7 : Remplacer tout X_i par X
8 : sinon
9 : si $(X_i)' \supset (X_j)'$ alors
10 : Remplacer X_j de $[P]$
11 : ajouter $X \times Y$ à $[P_i]$
12 : sinon
13 : si $(X_i)' \neq (X_j)'$ alors
14 : ajouter $X \times Y$ à $[P_i]$
15 : fin si
16 : fin si
17 : fin si
18 : fin si
19 : fin si
20 : retourner $[P]$

L'inconvénient majeur de l'algorithme *Charm* est qu'il nécessite un espace considérable de stockage. En effet, le fait de stocker les itemsets et leurs tidsets, ne fait qu'accroître la quantité de mémoire utilisée et ceci surtout pour de larges contextes épars.

Exemple. La figure suivante explicite les différentes étapes d'extraction des itemsets fréquents fermés par l'algorithme *Charm* pour *minsup*=2/5.

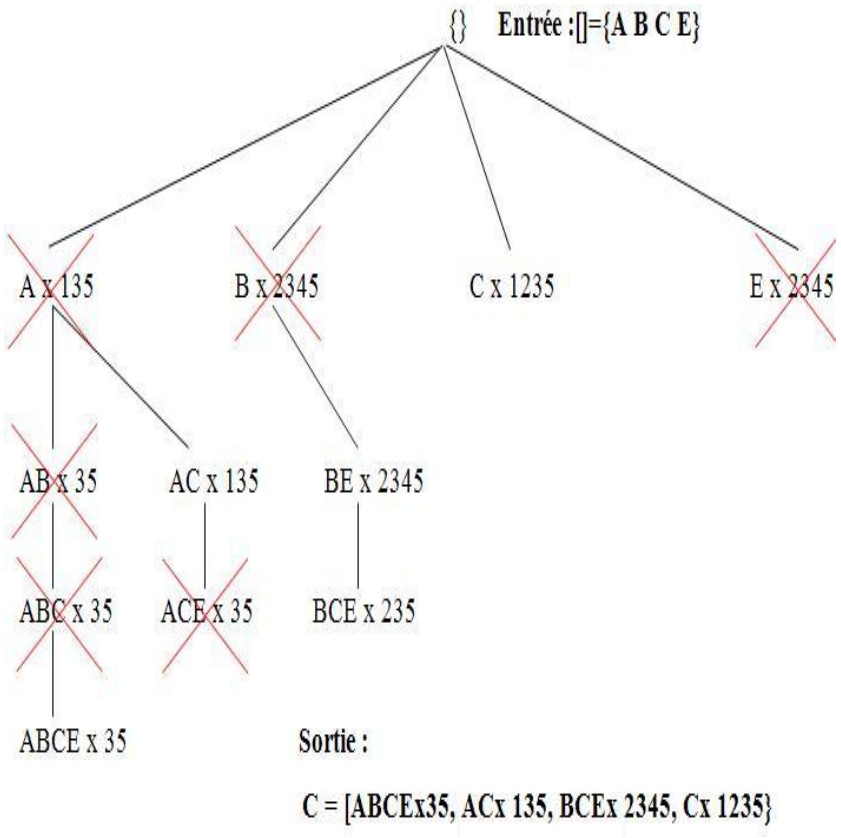

FIGURE 2.11: Etapes d'extraction des itemsets fermés fréquents par *Charm*

2.4.6 Extractions basé sur les itemsets maximaux

La recherche d'itemsets fréquents est une solution satisfaisante pour réduire le nombre des d'itemsets extraits [172], [115], [231], [174], [209], [230], [14].

Toutefois, ces performances se dégradent considérablement, lorsqu'il s'agit de très grandes bases de données fortement corrélées, ou les itemsets fréquents peuvent être très longs, comme dans le cas des bases de données biologiques ou boursières.

Ainsi, l'approche basée sur la découverte d'itemsets maximaux a pour but de ne considérer que l'ensemble des itemsets les plus longs parmi les itemsets fréquents. De ce fait, les maximaux sont des itemsets fréquents dont tout sur-ensemble est infréquent [19].

Plusieurs algorithmes ont été proposés pour la découverte des motifs fréquents maximaux (*MFI*) dans une base de données de transaction. L'objectif est d'éviter un parcours exponentiel du treillis des motifs candidats, en caractérisant au plus vite les grands motifs fréquents sans explorer leurs sous-ensembles.

Une bonne évaluation des approches d'extraction des motifs fréquents maximaux, que ce soit dans le contexte des bases de données denses et éparses, est donnée dans [97].

2.4.6.1 L'algorithme *Mafia*

L'algorithme *Mafia* [42], est un algorithme itératif d'extraction des itemsets maximaux qui effectue un parcours en profondeur du treillis. L'algorithme *Mafia* représente les données sous la forme de bitmaps verticaux qui se révèlent extrêmement efficaces. Il utilise aussi une stratégie effectuant des sauts dans l'espace de recherche pour limiter la taille de l'arbre.

Cette stratégie intègre un système de réordonnancement et trois principes d'élagage, qui peuvent être activés indépendamment les uns des autres, ce qui améliore considérablement les performances.

- *PEP (Parent Equivalence Pruning).* Si un itemset dans l'arbre a le même *support* que l'un de ses fils, alors il peut être élagué de l'arbre car il apparaitra dans la base de données seulement comme un sous-ensemble de son fils.
- *HUTMFI Superset Pruning.* Si l'union d'un itemset et du premier élément de sa queue est fréquente, alors le sous-arbre entier peut être élagué. Cette stratégie regarde dans la liste actuelle des *MFI* pour vérifier si l'ensemble tête queue est déjà dans cette liste.
- *FHUT (Frequent Head-Union-Tail).* Cette méthode d'élagage est identique à *HUTMFI* sauf qu'elle calcule le support de l'ensemble tête queue au lieu de chercher s'il est déjà dans la liste des itemsets fréquents maximaux (*MFI*). *FHUT* a des performances inférieures à *HUTMFI.*

Ainsi, l'application de la propriété que tous les sous-ensembles d'un *MFI* sont fréquents permet d'éviter de recalculer une nouvelle fois dès lors qu'il est déjà présent dans les *MFIs* connus. Il est donc moins coûteux de continuer à explorer directement à partir du sur-ensemble.

L'application d'un réordonnancement de la queue du nœud à explorer, permet d'améliorer considérablement le temps d'extraction des itemsets fréquents maximaux. Ceci consiste à proposer en premier les fils ayant le *support* le plus élevé. Ceci permet d'augmenter les chances de tomber sur un *MFI* à partir d'un fils ayant un *support* éloigné du *support* minimal, qu'à partir d'un fils ayant un *support* proche du support minimal. Dans le cas où le premier fils d'un nœud père et que tous ses fils sont fréquents, on supprime le nœud et on remonte dans le treillis.

2.4.6.2 L'algorithme *GenMax*

L'algorithme *GenMax* a été proposé par Gouda et Zaki [97] afin d'extraire les itemsets fréquents maximaux. *GenMax* est basé sur une recherche avec retour arrière combiné avec deux techniques d'optimisation permettant d'élaguer rapidement de larges portions de l'espace de recherche. La première technique, appelée progressive focusing, permet d'éliminer les itemsets non maximaux et la deuxième, appelée diffset propagation, permet de calculer rapidement la fréquence.

Les expérimentations menées dans [97] montrent que *GenMax* comparativement à d'autres algorithmes comme *Mafia* [42] et *MaxMiner* [19]présente les meilleurs résultats dans l'énumération d'ensemble d'itemsets maximaux étant donné qu'il ne nécessite pas une phase d'élagage après la phase d'extraction (l'élagage est effectué en même temps que l'extraction).

L'algorithme *GenMax* suppose un format vertical de la base de données pour chaque item, son *tidset* et l'ensemble des *tids* de toutes les transactions dans lesquelles cet item apparait. Il commence par calculer les 1-itemsets et les 2-itemsets fréquents. Cette information est utilisée dans l'ordonnancement des items dans la liste de combinaison initiale pour minimiser l'arbre constituant l'espace de recherche. Il utilise ensuite la technique progressive focusing de LFMI-backtrack combinée à la technique diffset propagation de FI-diffset-combine pour produire les itemsets fréquents maximaux exacts.

2.5 Bilan et discussion

L'approche d'extraction des itemsets fréquents ne peut être utilisée lorsque les données sont fortement corrélées. En effet, même pour des valeurs de *minsup* très élevés, le nombre d'itemsets fréquents extrait reste aussi très élevé. L'approche d'extraction des itemsets fermés fréquent permet de réduire le nombre d'itemsets extraits tout en gardant la qualité des résultats.

L'algorithme *Close* [172], [171] partage la même technique que l'algorithme *Apriori* [4]. Il privilégie une exploration par niveau de l'espace de recherche. Par contre, *Charm* [231] privilégie une exploration en profondeur d'abord de l'espace de recherche. Ainsi, l'algorithme *Charm* [231] explore simultanément l'espace de recherche des itemsets, ainsi que celui des identificateurs des transactions.

L'inconvénient majeur de l'algorithme *Close* [172], [171] et *Charm* [231] est qu'ils nécessitent un espace considérable de stockage. En effet, pour l'algorithme *Close* [172], [171], les auteurs supposent que l'ensemble des itemsets fermés fréquents de niveau k et l'ensemble des itemsets candidats de niveau $k+1$, utilisés dans la phase de construction d'un passage k, peuvent tenir en mémoire centrale.

Cependant, une telle quantité d'information est difficilement tenable en mémoire centrale, surtout pour des contextes denses et des valeurs de *support* très faibles. L'utilisation de l'algorithme *Charm* [231] nécessite le stockage de tous les itemsets et leurs *tidsets*. Ce que ne fait qu'accroître la quantité de mémoire utilisée.

2.6 Conclusion

Dans ce chapitre, nous avons présenté différentes approches data mining. Ces approches permettent d'extraire de la connaissance à partir d'un très grand volume de données. Néanmoins, elles doivent être adaptées pour mettre en place une stratégie efficace dans les contextes denses à grande volumétrie comme c'est le cas pour la sélection d'index et de vues matérialisées. A cet effet, une démarche d'extraction distribuée nous parait être une bonne solution pour combler les lacunes constatées.

Chapitre 3

Intelligence Artificielle Distribuée et Systèmes Multi Agents

3.1 Introduction

Etant donné la difficulté d'aborder la plupart des problèmes complexes de manière centralisée. Les recherches se sont concentrées sur l'intelligence artificielle distribuée (lAD) et les systèmes multi-agents (SMA). Ces travaux étudient la manière de répartir un problème sur un certain nombre d'entités coopérantes est la meilleure façon de coordonner ce comportement intelligent.

Ce chapitre présente une étude de l'univers multi-agents, tout en essayons de clarifier la terminologie du domaine en donnant aussi une vue d'ensemble sur son évolution.

3.2 L'intelligence artificielle

L'intelligence Artificielle (IA) est définie comme la science qui s'intéresse à la modélisation de comportement "intelligent".

Les problèmes relatifs à l'intelligence artificielle et la création de comportements "intelligents" à partir d'ordinateurs furent abordé à partir de 1960 [175]. Ainsi, *Turing* proposa un test consistant à dire qu'une machine est intelligente si on ne peut la distinguer d'un humain lors d'une conversation.

3.3 Définition d'un Agent

De nombreuses définitions du terme agent existent. Toutefois, il n'y a encore aucun consensus, entre les différents chercheurs, quant à une définition commune. Selon *Nwana* [160], les raisons de cette difficulté résident dans le fait que les chercheurs, dans le domaine des agents, ne sont pas à l'origine de ce terme.

En effet, le terme agent a été et continue d'être utilisé dans la vie de tous les jours par des personnes travaillant dans des domaines très différents. Par exemple, on parle d'agent de voyage, d'agent immobilier, d'agent d'assurance etc.

La deuxième raison est qu'étant donné que les agents peuvent prendre différentes formes physiques (robot ou agent logiciel) et qu'ils peuvent aussi jouer plusieurs rôles, ils peuvent ne pas être décrits par le même nom par les différents chercheurs.

Ainsi, on parle de « *knowbots* » (robots à base de connaissances), « *softbots* » (robots logiciel), *taskbots* (robots à base de tâche), « *userbots* » (robots pour utilisateur), robots, agents personnels, agents autonomes, assistants personnels, etc.

Pour Weiss [211], un agent est une "entité computationnelle", comme un programme informatique ou un robot, qui peut être vue comme percevant et agissant de façon autonome sur son environnement. Selon Sycara et Wooldridge [124], un agent est un système informatique, situé dans un environnement, qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu.

Wooldridge à son tour [219], adapte la définition donnée en [220] et déclare : 'Un agent est un système informatique qui se trouve dans certains l'environnement et qui est capable d'action autonome dans cet environnement afin de répondre à ses objectifs.

La définition que nous avons adoptée et qui semble couvrir les caractéristiques des agents que nous avons développés, est celle proposée par Ferber [74] et Foner [83].

Ces derniers s'accordent à définir un agent comme une entité autonome, capable de communiquer avec d'autres agents, ainsi que de percevoir (mais de manière limitée) et d'agir dans son environnement. Chaque agent effectue des actions spécifiques mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'il cherche à optimiser).

FIGURE 3.1: l'agent et son environnement

3.4 Caractéristiques des agents

Les chercheurs en intelligence artificielle s'accordent sur la nécessité de l'existence d'un certain nombre de caractéristiques pour un agent. Certaines de ces caractéristiques seront plus importantes que d'autres selon le type d'applications.

3.4.1 La Situation

Un agent est situé dans un environnement dans lequel il est capable de recevoir des entrées sensorielles. Il peut réagir et modifier cet environnement par les actions qu'il réalise.

3.4.2 L'autonomie

L'agent doit pouvoir agir sans intervention directe d'un humain (ou d'un autre agent). Il doit disposer d'un certain contrôle sur ses actions et de ses états internes [102].

3.4.3 Capacité à communiquer

L'agent doit pouvoir échanger des informations plus ou moins complexes avec d'autres agents. Cette communication se fait à l'aide de protocoles qui peuvent être de deux types :

3.4.3.1 La communication par partage d'information

Dans ce mode de communication, toute l'information sur le système, est centralisée dans une structure de données globale. Les agents y viennent pour lire et écrire afin de faire évoluer le système qui contient initialement les données du problème. Ce type de communication s'appelle «*tableau noir*» ou «*blackboard* ».

Ce mode de communication centralisé implique que les agents effectuent un grand nombre de requêtes sur un seul et unique site central. De plus, le fonctionnement global du système n'est pas purement multi-agents puisque le comportement d'un des agents est dépendant du contenu d'une base de connaissances commune.

3.4.3.2 La communication par envoie de message

Les systèmes multi-agents fondés sur la communication par messages se caractérisent par le fait que chaque agent possède une représentation propre et locale de l'environnement qui l'entoure. Chaque agent va alors interroger les autres agents sur cet environnement ou leur envoyer des informations sur sa propre perception des choses.

Cette communication permet de réaliser un véritable système multi-agents puisque chaque agent possède sa propre base de connaissance. Par contre, il est plus difficile d'assurer une convergence globale du système. De plus, dans un environnement distribué, la communication entre un grand nombre d'agents peut très vite amener à une saturation du réseau compte tenu de la grande quantité de messages échangés.

3.4.4 La capacité à coopérer

Les agents ont une connaissance plus ou moins précise des autres agents du système. Ils doivent pouvoir se représenter les compétences des autres agents, mais également les tâches que ces derniers sont en train de réaliser.

Ainsi, chaque problème doit être fractionné et réparti entre les différents agents qui produisent des solutions partielles qui doivent ensuite être fusionnées pour obtenir la solution globale.

Les modèles d'allocation des tâches peuvent être soit centralisés, un agent se charge alors de décomposer le problème et de le répartir, soit distribué, chaque agent est alors capable de décomposer le problème et de répartir les différentes tâches qui en résultent.

La collaboration entre agents peut parfois être de type conflictuel. Dans ce cas, les agents peuvent avoir des buts identiques, mais des vues divergentes voire totalement op-

posées. Les conflits entre les différents agents peuvent être résolus de deux manières différentes :

(1) grâce à un contrôle centralisé qui se charge, en dernier lieu, de prendre la décision. (2) par négociation entre les différents agents en conflits. Si le principe de la négociation semble plus intéressant pour faire émerger des solutions innovantes, ce choix pose d'énormes difficultés notamment une trop longue convergence de la négociation quand les conflits d'intérêts sont trop importants.

3.4.5 Capacité à raisonner et à réagir avec leur environnement

L'agent doit être capable de s'adapter à son environnement (qui peut être composé d'autres agents, du web en général ou des utilisateurs) et aux évolutions de celui-ci. Cette adaptation doit s'appuyer sur l'analyse de l'environnement extérieur des agents.

3.4.6 La mobilité

Les agents doivent pouvoir être multi-plate-forme et multi-architecture. Ils doivent pouvoir se déplacer sur le réseau où ils accomplissent des tâches sans que l'utilisateur ait le moindre contrôle sur celles-ci.

3.5 Les différentes catégories d'agents

Russel et Norvig [183] distinguent deux grandes catégories d'agents, à savoir : Les agents réactifs et Les agents cognitifs. A ces dernières, on retrouve dans la littérature les agents hybrides qui sont une synthèse de ces deux précédentes orientations.

3.5.1 Les agents réactifs

Les agents réactifs [38], [76], ont un simple comportement de type «stimulus/réponse». De ce fait, ils ne possèdent pas une représentation complète de leur environnement et ne sont pas capable de tenir compte de leurs actions passés. Cependant, ces agents ont la caractéristique de rapidité de réaction vue la simplicité de leurs architectures internes.

Les agents réactifs par leur coopération permettent de faire émerger un système global intelligent même s'ils ne sont pas intelligents individuellement [137], [71]. En effet, ces mécanismes simples de réactions aux événements peuvent faire émerger des comportements globaux intelligents correspondant aux objectifs poursuivis. Les sociétés d'insectes comme les fourmis, les termites sont les exemples les plus anciens d'agents réactifs. L'architecture la plus connue est celle de Rodney Brooks. On l'appelle aussi architecture de subsomption.

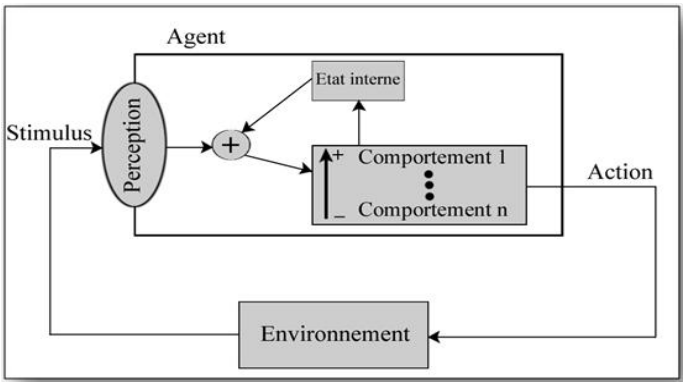

FIGURE 3.2: architecture de Subsomption (Rodney Brooks)

3.5.2 Les agents cognitifs

Les agents cognitifs sont plus évolués. Ils disposent d'une représentation globale de leur environnement et des autres agents avec lesquels ils communiquent.

Les systèmes d'agents cognitifs sont fondés sur la coopération d'agents capables à eux seuls d'effectuer des opérations complexes. Pour cela, chaque agent cognitif utilise pour raisonner, une base de connaissances comprenant l'ensemble des informations et de savoir-faire nécessaires à la réalisation de son objectif et à la gestion des interactions avec les autres agents et avec son environnement.

L'une des architectures les plus connues pour les agents cognitifs est celle inspirée de la théorie de Michael Bratmans [36]. Cette architecture s'appelle *BDI* pour croyance, désir et intention :

- Belief (croyances) : Ce sont les informations que l'agent possède sur l'environnement et sur les autres agents qui existent dans le même environnement. Les croyances peuvent être incorrectes, incomplètes ou incertaines et, à cause de cela, elles sont différentes des connaissances de l'agent, qui sont des informations toujours vraies. Les croyances peuvent changer au fur et à mesure que l'agent, par sa capacité de perception ou par l'interaction avec d'autres agents, recueille plus d'information.
- Désirs : Ils représentent les états de l'environnement (ou de l'agent lui-même), que l'agent souhaite atteindre. Un agent peut avoir des désirs contradictoires. Dans ce cas, il doit choisir parmi ses désirs un sous-ensemble qui soit consistant. Ce sous-ensemble est parfois identifié avec les buts de l'agent.
- Intentions : Ce sont les désirs que l'agent a décidé d'accomplir ou les actions qu'il a décidé de faire pour accomplir ses désirs [130]. Même si tous les désirs d'un agent sont consistants, l'agent peut ne pas être capable de les accomplir tous à la fois.

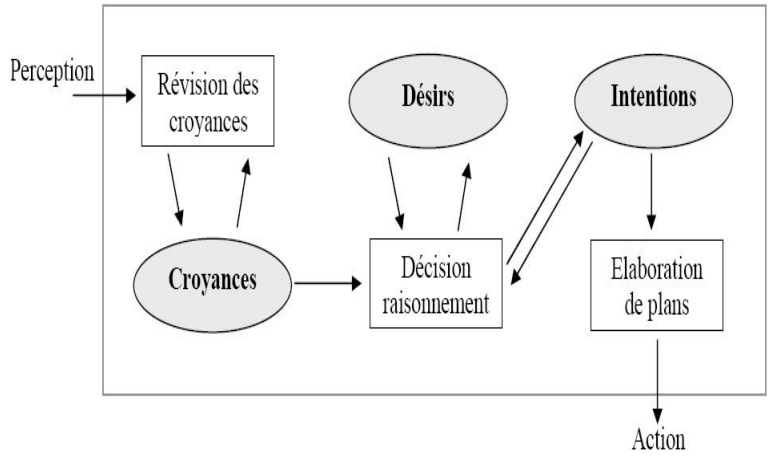

FIGURE 3.3: *Architecture d'un agent BDI*

3.5.3 Les agents hybrides

Les agents hybrides [78], [151] apportent une solution au problème de l'intégration des aspects réactifs et cognitifs en combinant ces deux propriétés généralement logées dans des modules différents et arrangées selon une hiérarchie.

Cette architecture modulaire fait de l'agent un système ouvert [103] en permettant l'intégration de capacités très hétérogènes en termes de programmation. Ceci lui permet d'adapter son comportement en temps réel à l'évolution de son univers. L'une des architectures hybrides les plus connues est celle du système InteRRaP.

FIGURE 3.4: Architecture InterRap

3.6 La notion d'environnement

Selon [28], un système multi-agents possède un environnement dans lequel plusieurs agents évoluent, communiquent, perçoivent et agissent. Cet environnement peut se limiter seulement aux messages échangés par des agents cognitifs. Pour un agent, on distingue l'environnement physique et social :

1. L'environnement physique correspond à tous les objets que l'agent peut percevoir ou sur lesquels il peut agir. On parle d'environnement physique essentiellement dans le cas d'agents situés ;
2. L'environnement social correspond aux autres agents avec lesquels un agent est en interaction par envoi de message par exemple. On parle d'environnement social surtout dans le cas d'agents communicants.

L'environnement peut être aussi classé, concernant sa nature et sa dynamique, selon les caractéristiques suivantes [182], [218], [144] :

1. · Accessible/Inaccessible : dans ce cas, un agent peut obtenir des informations exhaustives, précises, complètes et à jour au sujet de cet environnement. Plus un environnement est accessible, plus il est facile de construire des agents qui seront efficaces.
2. Déterministe/non-déterministe : c'est un environnement dans lequel une action a un effet connu et garanti. Aucun doute n'est possible sur les conséquences de cette action. Etant donné qu'un agent n'exécute des actions que dans le but de faire atteindre à l'environnement un état désiré, ce qui veut dire que dans un système non déterministe, l'agent doit concevoir la possibilité d'échec [219].
3. Statique/dynamique : un environnement statique n'est modifié que par les actions de l'agent. A l'opposé, l'agent n'est pas le seul acteur à effectuer des modifications dans un environnement dynamique.
4. Discret/continu : un environnement discret se caractérise par un nombre fixe et fini d'actions et de perceptions possibles.

La caractérisation de l'environnement avec celle des comportements individuels, permet de définir la dynamique d'un système multi-agents. Selon [212], les différents rôles de l'environnement sont de permettre la communication entre agents et d'être le support de leurs actions en définissant des règles. L'environnement permet aussi de prendre en charge l'activité propre des objets et des ressources présents en son sein.

Lorsque les agents sont réactifs, l'environnement détient une importance capitale car il est le médiateur de leurs interactions. En effet, comme ces agents ne peuvent communiquer directement entre eux, ils s'influencent mutuellement soit par leur position s'ils sont situés, soit par l'intermédiaire d'objets qu'ils perçoivent et modifient.

3.7 L'intelligence artificielle distribuée

Pour remédier aux insuffisances de l'approche d'intelligence artificielle, qui s'appuie sur une centralisation de l'expertise au sein d'un système unique même pour la résolution des problèmes complexes.

Un certain nombre de travaux ont proposé, depuis le début des années 70, de distribuer l'expertise sur un groupe d'agents devant être capables de travailler et d'agir dans un environnement commun et résoudre les conflits éventuels.

Cette nouvelle branche de l'*IA* qui est l'Intelligence Artificielle Distribuée (*lAD*), [30], [119], [77] peut être définie comme étant la science qui s'intéresse à la conception d'agents artificiels capables de s'organiser efficacement pour accomplir collectivement les fonctionnalités qui leur sont demandées.

3.8 Les systèmes multi-agents

Dans la plupart des situations réelles, l'agent n'est pas seul dans son environnement, il y a d'autres agents présents autour de lui. Ces agents vivant au même moment, partageant des ressources communes et communiquant et interagissant entre eux pour effectuer leurs tâches. De tels systèmes sont appelés «systèmes multi agents»[208], [101], [49], [219].

En général, les interactions sont mises en œuvre par un transfert d'informations entre agents ou entre l'environnement et les agents, soit par perception, soit par communication [49], [101].

Selon Ferber [75], un système multi-agents (*SMA*) est composé des éléments suivants :

1. *Un environnement E.* Un espace disposant généralement d'une métrique.
2. Un ensemble d'objets O qui sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.
3. Un ensemble A d'agents qui sont des objets particuliers ($A \subseteq O$), lesquels représentent les entités actives du système.
4. Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.
5. Un ensemble d'opérations Op, permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O.
6. Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, qu'on appellera les lois de l'univers.

3.9 Caractéristiques principales des systèmes multi-agents

Selon [124], les systèmes multi-agents possèdent principalement les caractéristiques suivantes :

- chaque agent a des informations ou des capacités de résolution de problèmes incomplètes. Donc chaque agent a un point de vue limité,
- il n'y a pas de contrôle global du système (les agents doivent être autonomes),
- les données sont décentralisées,
- les calculs sont asynchrones,
- les messages doivent être mutuellement compréhensibles par les différents agents.

Ainsi, les systèmes multi-agents uniquement cognitifs sont constitués d'un nombre d'agents assez faible. Ils réclament des ressources plus importantes que les systèmes d'agents réactifs.

La convergence du système vers un état décisionnel stable n'est pas non plus assurée par l'utilisation de ce type d'agents. Cependant, ils permettent de résoudre des problèmes plus complexes et nécessitant une plus grande abstraction.

3.10 Utilité des systèmes multi-agents

Les systèmes multi-agents possèdent en premier lieu les avantages liés à la résolution distribuée et concurrente de problèmes [49], à savoir :

3.10.1 La modularité

Elle permet, aux systèmes multi agents d'être facilement extensibles. Ceci rend la programmation plus simple. En effet il est plus facile d'ajouter de nouveaux agents à un système multi-agents que d'ajouter de nouvelles capacités à un système mono modulaire.

3.10.2 La vitesse

Elle est principalement due au parallélisme, car plusieurs agents peuvent travailler en même temps pour la résolution d'un problème.

3.10.3 La fiabilité

La fiabilité d'un système multi agents peut être atteinte, dans la mesure où le contrôle et les responsabilités étant partagées entre les différents agents, le système peut tolérer la défaillance d'un ou de plusieurs agents. Si une seule entité contrôle tout, alors une seule défaillance de cette entité fera en sorte que tout le système tombera en panne.

3.11 Les interactions dans les systèmes multi-agents

La notion d'interaction est au centre de la problématique des systèmes multi-agents. Une interaction est une mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques.

Les interactions s'expriment ainsi à partir d'une série d'actions dont les conséquences exercent en retour une influence sur le comportement futur des agents [75]. Les systèmes Multi-Agents s'intéressent aux comportements collectifs produits par ces interactions selon plusieurs schémas :

- La communication
- La planification
- La négociation
- L'organisation
- La coopération

3.11.1 La communication

La communication permet d'élargir la perception de l'environnement par les agents, puisqu'elle leur permet de bénéficier de celles des autres agents et de leurs connaissances par l'échange des informations et des demandes de services [26].

Les agents communiquent entre eux à l'aide d'un langage en utilisant des protocoles. Ceci permet à un agent d'agir sur un autre agent en lui fournissant des informations qui auront pour conséquence la remise en question de son comportement.

3.11.2 La planification

La planification multi-agent s'intéresse à la coordination distribuée des actions entre agents. Il existe deux grands modes d'organisation de la planification multi-agent :

3.11.2.1 La planification centralisée

Dans ce mode d'organisation, chaque agent produit un plan de résolution et un agent central ayant une vue globale du système coordonne ces différents plans afin d'éviter les conflits.

3.11.2.2 La planification distribuée

Dans ce mode d'organisation, chaque agent à son propre plan partiel. Le but est de permettre aux agents de co-construire un plan de résolution en tenant compte des compétences de chaque agent au fur et à mesure de la co-construction.

La négociation est aussi une forme de coopération, car c'est un mécanisme par lequel les agents peuvent arriver à un accord commun quand ils doivent coordonner leurs actions pour résoudre un problème.

3.11.3 L'organisation

Un système multi-agents est un système constitué d'un ensemble d'agents qui interagissent avec leur environnement dans le but de faire adopter au système un comportement global. La notion d'organisation dans un système multi-agents peut être appréhendée donc avec deux approches.

La première consiste à considérer que l'organisation des agents émerge de leur comportement. Dans ce cas la société d'agents adopte une dynamique collective, alors qu'aucun agent de la société n'est capable de "penser" cette dynamique [67].

La seconde s'intéresse à la manière dont les activités sont structurées [111], [207]. On se concentre donc sur la description des activités des agents, car l'organisation des agents est préétablie.

3.12 Exemples de protocoles d'interaction

Nous décrivons dans cette section certains protocoles d'interaction les plus utilisés par la communauté multi-agents.

3.12.1 Le protocole Contract Net

Ce protocole fut l'une des premières approches utilisées dans les systèmes multi-agents pour résoudre le problème d'allocation des tâches [192].

Dans ce protocole, les agents peuvent prendre deux rôles : manager et contractant. Le manager est responsable de la surveillance de l'exécution d'une tâche et du traitement des résultats de cette exécution.

Le manager commence par décomposer la tâche en plusieurs sous-tâches. Par la suite, Le manager annonce chaque sous-tâche aux agents contractants. Après évaluation, les agents contractants envoient au manager des soumissions qui indiquent leurs capacités à réaliser la tâche. Le manager évalue toutes les propositions qu'il a reçues et alloue la tâche à l'agent qui a fait la meilleure proposition.

3.12.2 Les protocoles d'enchère

Nous retrouvons dans la littérature plusieurs types d'enchères, mais on se limite dans cette section à une brève présentation des types d'enchères les plus utilisés dans les systèmes multi-agents.

3.12.2.1 Enchère anglaise

Dans l'enchère anglaise [81], tous les agents ont connaissances du déroulement de l'enchère. L'initiateur commence l'enchère en annonçant un premier prix initialement inférieur à la valeur estimée du produit (le prix minimal pour lequel il est d'accord pour vendre l'objet). Chaque participant annonce publiquement son offre, en plusieurs tours successifs. A chaque fois qu'un participant se manifeste, l'initiateur annonce une nouvelle proposition de prix égale au dernier prix incrémenté d'une valeur appelée pas d'incrémentation. Quand aucun participant ne veut plus augmenter son offre, l'enchère s'arrête et le participant ayant fait la plus grande offre gagne l'objet au prix de son offre.

3.12.2.2 Enchère hollandaise

Dans ce type d'enchère [81], aucun agent n'a connaissance de la mise des autres agents. Lorsque l'agent remporte l'enchère (en ayant proposé la somme la plus élevée), il remporte le produit mais au prix de la seconde mise, i.e celle se trouvant juste au-dessous de la mise gagnante.

L'initiateur commence par annoncer une valeur très élevée par rapport à la valeur estimée du produit. Puis il commence par baisser le prix jusqu'à ce qu'un participant signale son acceptation du prix proposé. L'initiateur possède une estimation de la valeur du produit (prix de réserve) au-dessous de laquelle il ne vend pas le produit. Si l'enchère atteint le prix de réserve sans qu'il y ait d'acheteurs, l'enchère se termine.

3.12.2.3 Enchère Vikrey

Ce type d'enchères [206] portent le nom de l'économiste américain William Vickrey. Comme les enchères anglaises, elles permettent de vendre un article unique.

Chaque participant soumet une offre sans savoir les offres des autres, dans un seul tour. Le participant qui a fait l'offre la plus grande gagne mais il doit payer le prix de la deuxième plus grande offre. William Vickrey a démontré que ce type d'enchères incitent les enchérisseurs à parier sur la valeur réelle qu'ils attribuent au lot.

3.13 La coopération dans les systèmes multi-agents

Charles Lenay [141] définit la coopération comme "un type de dynamique collective qui aboutit à un état émergent bénéfique pour la population". Ainsi, grâce à la coopération, chaque agent qui ne possède qu'une vue partielle du système auquel il appartient, peut réaliser plus de tâches que s'il avait travaillé seul de manière individualiste.

Le concept de coopération peut être utilisé pour signifier la coopération pour la résolution du problème auquel est confronté le système, ou bien pour indiquer la coopération entre les agents, pour que le système ait un fonctionnement optimal.

Le Petit Robert définit la coopération comme "action de participer à une œuvre commune". Demazeau et Müller [63] parlent de coopération pour une tâche locale. Ainsi un agent a besoin de coopérer avec autrui parce qu'il n'est pas capable de l'accomplir par lui-même ou parce que les autres peuvent l'accomplir de manière plus efficace que lui (dans un intervalle de temps plus court).

A la différence de la résolution connexionniste ou par les systèmes neuronaux, dans lequel les nœuds individuels sont très simples et n'ont pas une véritable expertise. Les agents ont généralement l'expertise qui est liée, mais distinctes et qui doit être combinée pour résoudre les problèmes. Ces travaux communs sont nécessaires du fait des dépendances entre les actions des agents, la nécessité de répondre à des contraintes globales et parce qu'aucun agent ne possède une compétence suffisante, les ressources ou les informations pour résoudre le problème tout seul.

Les activités entreprises par les agents individuellement sont liées soit parce que les décisions locales prises par un agent ont un impact sur les décisions d'autres membres de la communauté (par exemple lors de la construction d'une maison, les décisions concernant la taille et la localisation des chambres, a des impacts sur le câblage et la plomberie), ou en raison de la possibilité d'interactions dangereuses entre les agents (par exemple, deux robots mobiles peuvent entrer en collision en tentant de passer par une étroite sortie).

3.13.1 Les formes de coopération

Jacques Ferber [75] ainsi que Christian Brassac et Sylvie Pesty [35] s'entendent sur le fait que, selon le niveau d'observation où l'on se place, la coopération est considérée suivant deux formes.

3.13.1.1 *la coopération comme une attitude intentionnelle des agents*

Dans ce cas, c'est les agents qui décident de travailler en commun après avoir identifié et adopté un but commun. La coopération résulte de la prise de conscience de l'existence

d'un but commun et de l'engagement de chacun des agents dans cette action commune, c'est-à-dire reconnaissent que les autres agents sont engagés dans le même but [89].

3.13.1.2 *la coopération comme une interprétation des activités d'un ensemble d'agents.*

Dans ce cas, la coopération est considérée comme une qualification de l'activité d'un ensemble d'agents par un observateur extérieur qui n'aurait pas accès aux états mentaux des agents. Pour un certain nombre d'auteurs et notamment [70], [33], le comportement des agents est qualifié de coopératif, lorsqu'on observe un certain nombre de phénomènes que l'on utilise comme des indices d'une activité de coopération [68].

Ils considèrent que plusieurs agents coopèrent, pour atteindre un objectif commun, si l'une des deux conditions est vérifiée :

a) l'ajout d'un nouvel agent accroît différentiellement les performances du groupe. Ce critère est vrai puisque si les agents coopèrent, un nouvel agent est considéré comme une aide pour l'ensemble du groupe.

b) il existe des conflits potentiels ou actuels, l'action des agents sert à éviter ou à sortir de ces derniers.

3.13.2 Les méthodes de coopération

Selon Ferber [75], il existe six méthodes de coopération :

3.13.2.1 Le regroupement

Pour les agents, le regroupement consiste à se rapprocher pour former un bloc homogène. Cette situation de coopération résulte de leurs obligation d'agir pour satisfaire leurs objectifs en tenant compte des contraintes provenant des ressources plus ou moins limitées dont ils disposent et de leurs compétences individuelles.

3.13.2.2 La multiplication

Elle signifie que la simple augmentation quantitative des individus dans un système donné permet de prévenir les défaillances individuelles possibles.

3.13.2.3 La spécialisation

C'est le processus par lequel des agents deviennent de plus en plus adaptés à leurs tâches.

3.13.2.4 La collaboration par partage de tâches et de ressources

La collaboration est l'ensemble des techniques permettant de répartir des tâches, des connaissances et des ressources entre les agents afin de résoudre un problème commun.

Cette répartition s'effectue par des mécanismes d'offre et de demande. Elle est réalisée, soit par un agent coordinateur qui centralise les offres et les demandes, soit de manière distribuée.

3.13.2.5 La coordination d'actions

La coordination d'actions est la propriété d'un système d'agents accomplissant une activité en environnement partagé. Elle est nécessaire pour les raisons suivantes :

- les agents ont besoin d'informations et de résultats que seuls d'autres agents peuvent fournir.
- les ressources sont limitées.
- les coûts sont optimisés. - les agents peuvent accomplir leur travail et satisfaire leurs objectifs.
- La résolution de conflits par arbitrage et négociation : L'arbitrage consiste à rédiger des lois et des règles de comportements auxquelles les agents devront se plier.

3.14 Les protocoles de communication dans un système Multi Agents

Il existe principalement trois modes de communication dans un système multi agents :

3.14.1 La communication par mémoire partagée

Le modèle de blackboard [117] définit une architecture qui organise la résolution de problèmes par coopération de plusieurs agents, appelés sources de connaissances, autour d'une base de données partagée appelée blackboard.

Chaque agent vient lire ou écrire sur le blackboard. Ce dernier lui permettant également de chercher de nouvelles informations et de les filtrer suivant le rôle qu'il a pour résoudre un sous-problème particulier.

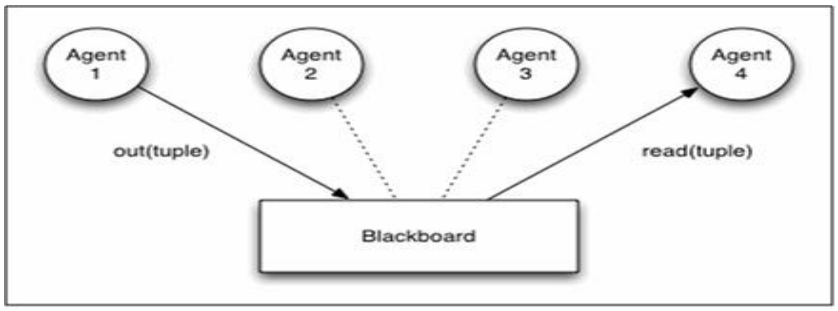

FIGURE 3.5: Communication par blackboard

3.14.2 La communication par messages (Acteurs)

Elle est représentée par l'envoi de messages asynchrones suivant les protocoles de communication définis. Les connaissances et les mécanismes de traitement pour la résolution d'un problème sont distribués dans les agents. La communication s'établit alors point par point ou par diffusion (" broadcast").

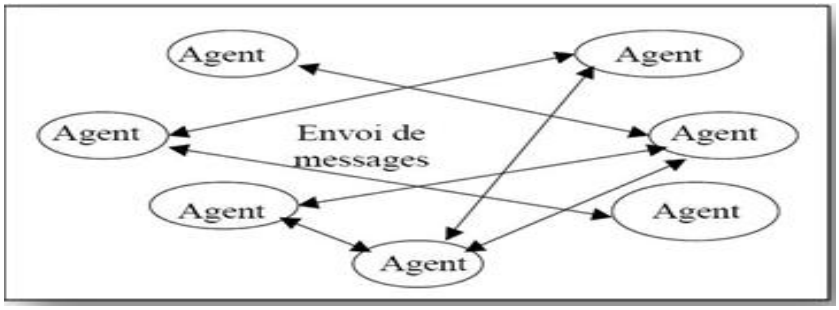

FIGURE 3.6: Communication par Messages

Enfin, les agents peuvent communiquer entre eux par l'intermédiaire d'un autre agent. Dans ce cas la communication entre les agents du système et l'agent intermédiaire se fait par le biais de messages directs.

On trouve ce genre de communication dans le protocole *ContractNet* [82] où on utilise le concept de négociation pour adjuger des contrats.

3.15 Langages de communication entre agents

Contrairement aux techniques *RPC* (Remote Procedure Call) [150], *CORBA* (Common Object Resource Broker Architecture) [164] et *RMI* (Remote Method Invocation) [147] qui essayaient de faire communiquer des programmes informatiques. Ceci les cantonnent au cadre de la programmation impérative ou procédurale : à chaque instant, le programme sait quelle procédure appeler.

Les langages de communication entre agents communément appelé *ACL* (Agent Communication Language), manipulent des propositions, des règles et des actions au lieu de simples objets sans sémantique associée.

Un message décrit un état dans un langage déclaratif, au lieu d'une simple invocation de méthode ou d'un simple appel de procédure.

Les types de messages des langages de communication entre agents sont appelés des actes de langage. Ces actes de langages sont issus des travaux de John Searle [189], qui a travaillé sur la conceptualisation du langage humain.

De ce fait un *ACL* désigne d'une part les langages de contenu qui permettent de structurer les connaissances échangées et d'autre part les langages de contenant qui définissent l'enveloppe des messages sans s'intéresser à leurs contenus [130]. Notant aussi que l'utilisation d'un langage de communication implique que tous les agents comprennent le vocabulaire sous tous ses aspects :

- la syntaxe, qui précise le mode de structuration des symboles.
- la pragmatique, pour pouvoir interpréter les symboles.
- l'ontologie, pour pouvoir utiliser les mêmes mots d'un vocabulaire commun.

3.15.1 Knowledge Query and Manipulation Language (*KQML*)

Le langage *KQML* développé au sein du groupe de travail «External Interfaces Group» du projet «Knowledge Sharing Effort, est basé sur les actes de langage dont le but est de permettre aux agents de coopérer. C'est un langage de communication et un protocole entièrement indépendant de la syntaxe du message. Conceptuellement, le langage *KQML* être considéré comme composé de trois couches :

1. La couche de contenu, est le message en lui-même. Il peut être exprimé avec n'importe quel langage de représentation (*KIF*, *FIPA-SL*, *RDF*).
2. La couche de communication comporte les paramètres de communication essentielles telles que les identités des expéditeurs et des destinataires.
3. La couche message est le noyau de *KQML*. Elle permet d'identifier la performative associée au contenu, c'est-à-dire spécifié si le message est une affirmation ou une question par exemple.

KQML s'appuie sur la définition de deux ensembles :

1. Un ensemble d'actes de communication primitifs, auquel s'ajoutent les autres actes de communication pouvant être obtenus par la composition de ces actes de base ;
2. Un ensemble de messages prédéfinis que tous les agents peuvent comprendre.

Bien que de nombreuses applications ont été réalisées avec *KQML*, le langage n'est pas vraiment devenu un standard. La complexité d'implémentation d'un système qui réaliserait les objectifs de *KQML* sont un frein à son développement. De plus, le manque de consensus autour d'un outil ou d'un langage pour la gestion des ontologies ne facilite pas la tâche.

3.15.2 Langage de communication *FIPA-ACL*

FIPA (Fondation for Intellegent Physical Agents) est une association internationale sans but lucratif, travaillant pour produire des spécifications pour des technologies agents.

Elle a comme objectif d'établir des standards afin de favoriser l'interopérabilité des applications, des services et des équipements informatiques. Elle vise à produire des normes pour assurer une conception uniforme des agents indépendamment de la plateforme.

En 1999, la *FIPA* a défini un *FIPA-ACL* [84] dans le but de corriger les défauts du *KQML :* ses performatifs sont trop nombreux, donc redondants et de plus ils ne couvrent pas en entier le champ des performatifs envisageables.

Le langage de communication *FIPA-ACL* est lui aussi basé sur la théorie des actes du langage, mais il ne comporte que 22 performatifs. La sémantique des performatifs est décrite dans un langage spécialisé, le langage *SL* pour Semantic Language [84].

Cette norme n'impose pas de mécanisme spécifique pour le transport interne de messages, car les agents pourraient s'exécuter sur des plateformes différentes et utiliser des technologies différentes.

Les 22 actes communicatifs de *FIPA-ACL* peuvent être groupés selon leurs fonctionnalités de la façon suivante :

– passage d'information : inform*, inform-if (macro act), inform-ref (macro act),confirm*, disconfirm* ;

- réquisition d'information : query-if, query-ref, subscribe ;
- négociation : accept-proposal, cfp, propose, reject-proposal ;
- distribution de tâches (ou exécution d'une action) : request*, request-when,request-whenever, agree, cancel, refuse.
- manipulation des erreurs : failure, not-understood.

3.16 Les plates-formes de développement des systèmes multi-agents

Une plate-forme de développement des systèmes multi-agents, correspond à une infrastructure logiciel utilisée comme environnement pour le déploiement et l'exécution d'un ensemble d'agents. Elle doit permettre de créer exécuter et supprimer des agents en plus du fait qu'elle doit agir en tant qu'un médiateur entre le système d'exploitation et les applications (agents) tournant dessus.

3.16.1 La plate-forme *MADKIT* (Multi-Agents Developpement Kit)

MADKIT [110] est une plate-forme multi-agents écrite en Java, développée par le Laboratoire d'Informatique, de Robotique et de Micro électronique de l'Université Montpellier II (LIRMM).

MadKit est une plateforme modulaire conçue selon le modèle d'organisation Alaadin [75] dans lequel des agents sont situés dans des groupes et jouent des rôles.

3.16.2 La plateforme *JADE*

JADE est une plate-forme de développement d'agents, gratuite et open source qui est conforme aux spécifications *FIPA97* [81]. Elle est basée sur la coexistence de plusieurs Machines Virtuelles Java qui communiquent par la méthode *RMI* (Remote Method Invocation) [147].

Chaque machine virtuelle fournit un environnement complet pour l'exécution simultanée d'un ensemble d'agents. Elle règle le cycle de vie des agents en les créant, les suspendant, les reprenant et les détruisant. De plus, elle traite tous les aspects de la communication : répartition des messages *ACL* reçus et dépôt des messages dans les files de messages privées des agents. Un récipient spécial contient les agents de gestion et représente la plate-forme toute entière pour le monde extérieur.

Une interface graphique utilisateur qui a été implémentée comme un agent, appelé *RMA* (Remote Monitoring Agent) permet de contrôler et superviser les états des agents, par exemple arrêter et remettre en marche un agent. Cette interface permet aussi de créer et de commencer l'exécution d'un agent sur un hôte éloigné, à condition qu'un réceptacle d'agents s'exécute déjà sur cet hôte.

3.16.3 AgentBuilder

AgentBuilder est une suite d'outils intégrés développée en *JAVA* par Reticular Systems Inc. *AgentBuilder* permet de construire des agents intelligents. Elle est basée sur les modèles *BDI* Agent [190] et Placa [201] et *KQML* est utilisé comme langage de communication entre les agents.

AgentBuilder contient deux composants (voir figure ci-dessous) : le toolkit et le système d'exécution (Runtime). Le toolkit contient des outils pour gérer le processus de développement des agents, analyser le domaine d'influence d'un agent, concevoir l'environnement de communication sur le réseau, définir le comportement individuel de chaque agent, débuguer et tester les agents. Le runtime fournit l'environnement d'exécution des agents et la combinaison du programme d'agent et du moteur d'agent permet de créer un agent exécutable. La méthodologie globale est documentée dans l'AgentBuilder User's Guide [198].

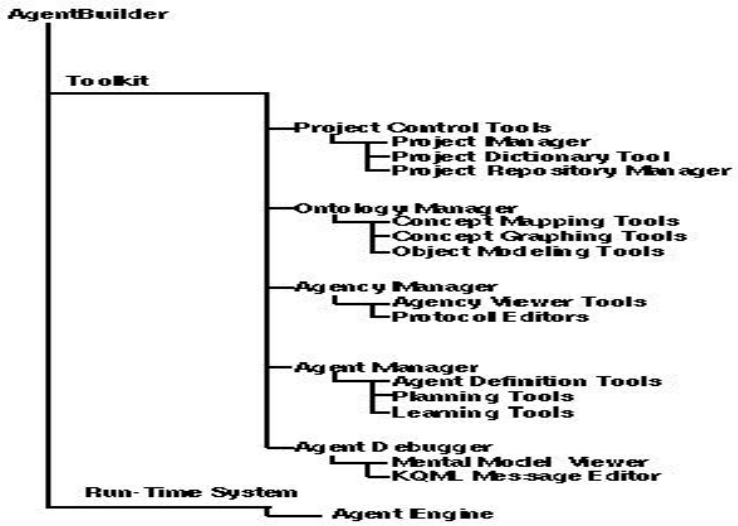

FIGURE 3.7: Les relations entre les principaux composants d'AgentBuilder

3.16.4 Jack

Jack est une plate-forme de développement orienté agent *BDI*, implémenté en *Java* [43]. Il est développé par la société australienne Agent Oriented Software Pty. Ltd.,. *JACK* est constitué d'un environnement de programmation graphique *JDE* (Jack Development Environment), du compilateur *JAL* (Jack Agent Language) qui traduit les programmes écrits en langage d'agent et les convertit en Java pur et de la librairie des classes permettant l'exécution des agents, appelée Jack Agent Kernel.

3.16.5 La plateforme ZEUS

Zeus est une plateforme développée par l'Agent Research Program du British Telecom Intelligent System Research Laborator pour la construction rapide d'applications à base d'agents collaboratifs [58].

Elle offre un environnement intégré pour capturer la spécification d'agent fournie par l'utilisateur et génère automatiquement le code source exécutable de l'agent spécifié. Zeus a été développé pour deux raisons fondamentales :

- Le besoin croissant d'un outil générique, personnalisable et fort industriellement pour construire des systèmes multi-agents distribués.
- Faciliter l'ingénierie d'agents et accélérer le processus de développement en encourageant la réutilisation du code et la standardisation de la technologie d'agents.

3.17 Méthodologie de conception et de modélisation Multi-agents

3.17.1 Australian Artificial Intelligence Institute Methodology (AAII)

AAII a été notamment développée par Kinny et al., [132] pour la gestion du trafic aérien. *AAII* est dédiée à la conception d'agents BDI impliqués dans une organisation et occupant des rôles précis.

3.17.2 AGR (Agent-Groupe-Rôle)

Ce modèle proposé par Jaques Ferber est fondé sur la notion de service, qui permet aux agents d'exprimer leurs besoins et compétences au travers de descriptions des rôles faciles à publier et rechercher.

Il est basé sur un concept simple : on suppose qu'un agent possède un ou plusieurs rôles au sein d'un ou plusieurs groupes. Un rôle est spécifique à un seul groupe. Il est une abstraction de la fonction, du service ou tout simplement l'identificateur d'un agent au sein d'un groupe.

La communication entre les agents, est possible grâce aux rôles qu'ils assument et le contrôle sur les communications est effectué par le groupe.

Dans *AGR*, une organisation est la relation structurelle entre les rôles et l'interaction n'est pas explicitement représentée ou utilisée dans le système. Les étapes principales de cette méthode sont :

- L'analyse : identifier les fonctions du système et les dépendances au sein de communautés identifiées. Il convient donc de définir quels sont les mécanismes de coordination et d'interaction entre les entités d'analyse.

- La conception : à cette étape se fait l'identification des groupes et des rôles dans des diagrammes de structures organisationnelles.

- La réalisation : qui commence par le choix de l'architecture d'agents suivie de la gestion des entités du domaine qui permet d'implanter le système à partir d'organisations concrètes.

Enfin, la méthode ne fournit pas de principes permettant de faciliter la décomposition un système complexe.

3.17.3 GAIA

Cette méthodologie part de l'hypothèse qu'un système multi-agents doit être vu comme une organisation "computationnelle" basé sur les interactions entre les différents rôles présents dans le système. Elle se compose de deux phases principales :

- L'analyse : qui produit deux modèles : le modèle de rôles et le modèle d'interactions,
- La conception : qui transforme les modèles abstraits de la phase d'analyse en modèles moins abstraits. L'étape de conception se base sur les trois modèles : Le modèle d'agent, le modèle de service et le modèle de relation. Un rôle dans *Gaia* est définit par quatre attributs :

• Les responsabilités : ce sont "les invariants" du rôle, qui se décomposent en deux catégories :

1. vivacité : un comportement a déclenché selon certaines conditions.
2. sûreté : qui consiste à s'assurer que certaines conditions restent valides.

• Les permissions : les droits accordés à ce rôle. Ces droits concernent principalement les droits d'accès à l'information.

• Les activités : qui correspondent au comportement "privé" de l'agent, déclenchés sans que l'agent soit en interaction.

• Les protocoles : ils définissent la structure des interactions entre les rôles.

Cette méthodologie est intéressante sur bien des points, mais reste trop générale pour pouvoir facilement passer de la phase de conception du système à sa réalisation. La figure ci-dessous décrit les modèles de *GAIA* présentés dans [221].

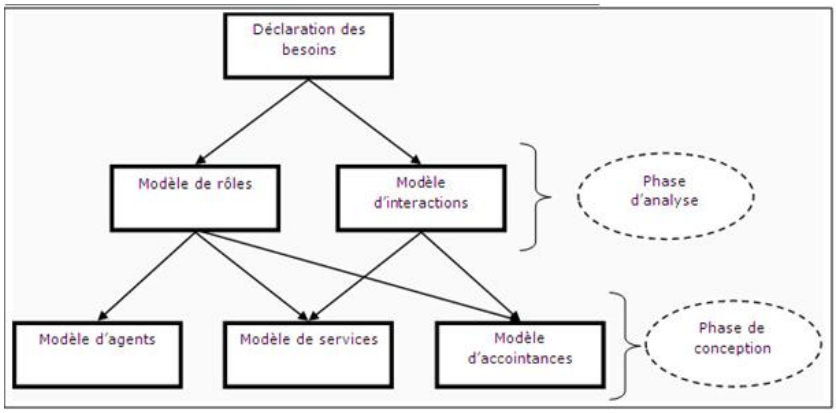

FIGURE 3.8: Les modèles de GAIA

3.17.4 La méthodologie VOYELLES « AEIO »

Cette méthodologie proposée par Demazeau [62], repose sur quatre concepts identifiés par les quatre voyelles : A pour Agents, E pour Environnements, I pour Interactions et O pour Organisations.

Chaque voyelle correspond à un ensemble de modèles qui devront être combinés pour aboutir à la réalisation du système multi-agent :

- *Agents* : concernent les modèles (ou les architectures) utilisés pour la partie active de l'agent.
- *Environnements* : sont les milieux dans lesquels sont plongés les agents, ils permettent de définir l'ensemble des capacités de perception et d'actions des agents.
- *Interactions* : concernent les infrastructures, les langages et les protocoles d'interactions entre agents, depuis de simples interactions physiques à des interactions langagières par actes de langage.
- *Organisations* : structurent les agents en groupes, hiérarchies, relations etc. Un autre concept a été ajouté à cette décomposition est qui celui de l'utilisateur qui consiste à définir les utilisateurs du système.

La méthodologie « AEIO » se décompose en trois phases qui sont :

3.17.4.1 La phase d'analyse

Elle permet d'identifier et de décomposer le problème en quatre composantes fondamentales : Agents, Environnement, Interactions et Organisation, pour dégager l'architecture générale du système.

3.17.4.2 La phase de conception

Elle permet de choisir les modèles opérationnels des composantes pour aboutir à la spécification du fonctionnement global du système exprimé sous forme de diagrammes de comportement.

Les modèles d'agents ou architectures d'agents vont des modèles simples comme les automates à états finis (agents réactifs) aux modèles plus complexes comme les systèmes à base de connaissances (agents cognitifs).

Les modèles d'environnement dépendent du domaine d'application. Ils sont généralement spatialisés et dotés d'une métrique.

Les structures et langages d'interaction vont des modèles issus de la physique, comme les modèles à base de forces, à des interactions de haut niveau basées sur la théorie des actes de langages. Enfin, les modèles d'organisations vont des modèles biologiques jusqu'aux modèles inspirés par les lois sociales.

3.17.4.3 La phase de programmation : (ou implémentation)

Elle consiste en l'instanciation des modèles, en utilisant des plates-formes et des langages choisis. Le système implémenté, peut alors être exécuté, évalué (test et validation) et repensé en cas d'inadéquation avec les besoins exprimés par le type de problème et le domaine d'application [176].

3.17.5 Agent UML (*AUML*)

UML (Unified Modeling Language) est un langage de modélisation issue principalement des travaux de *Booch*, *Jacobson* et *Rumbaugh* [11]. En peu de temps *UML* est devenu un standard dans la plupart des secteurs de l'informatique. Cependant, un certain nombres

de limitations du langage *UML* pour modéliser les agents peuvent être constatés. On peut citer :

UML	Description	Limitations
diagrammes d'interaction	Définit le comportement de groupes d'objets	Ne convient pas pour décrire les types d'interactions complexes entre les agents
diagrammes d'états	Définit tous les états possibles d'un objet	Convient pour un seul objet mais pas pour un groupe d'objets coopérants

TABLE 3.1: Quelques limitations du langage UML pour modéliser les agents

Etant donné la technologie pointue et les particularités propres à la modélisation multi agents, il était indispensable d'enrichir *UML* par des formalismes permettant la vérification et la validation des modèles qui représentent la partie dynamique du système. Cette dynamique regroupe l'expression de toutes les interactions pouvant avoir lieu entre les agents ainsi que la communication.

Dans ce cadre, le langage *AUML* (Agent-UML) a été proposé par Bauer, Odell et al., en 1999 comme étant une extension de la notation *UML* pour la modélisation d'agents [18], [17], [162], [163].

Les diagrammes du langage de modélisation *AUML* permettent d'exprimer les caractéristiques spécifiques aux agents (diagramme de classe d'agent) ainsi que les processus de communication (diagrammes de séquence) qui impliquent des structures conditionnelles assez lourdes à représenter avec *UML* [60].

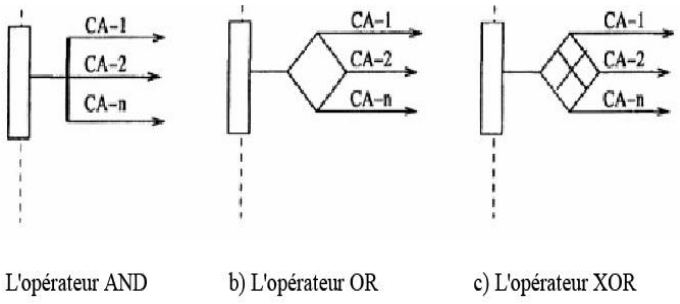

FIGURE 3.9: Types de connecteurs dans AUML

Les diagrammes de séquence montrent l'échange de messages entre les agents dans le temps en utilisant les protocoles de communication. Ces diagrammes sont composés d'un axe vertical représentant le temps et un autre horizontal représentant différents agents ou rôles d'agents.

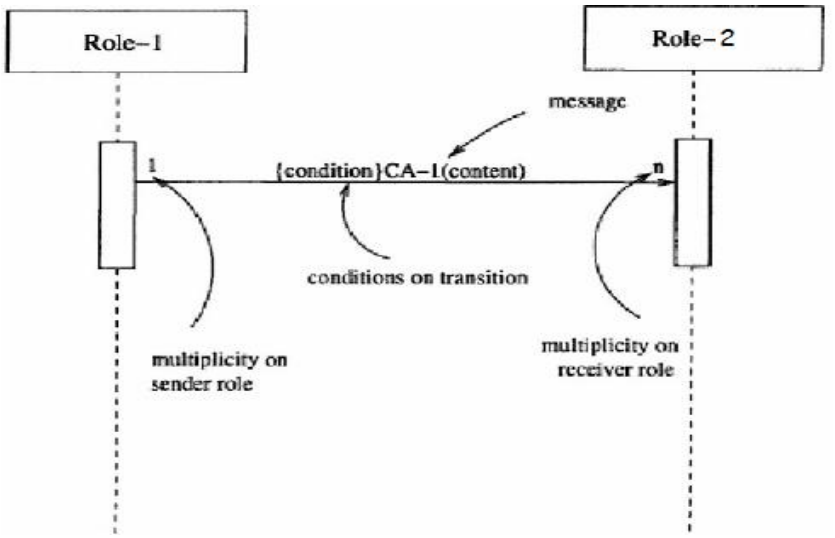

FIGURE 3.10: Diagramme de séquence dans AUML

3.18 Conclusion

Dans ce chapitre, nous avons présenté une synthèse de la littérature liée à la conception d'un système multi agents. Ce thème de recherche qui fait intervenir plusieurs domaines de recherche tel que la biologie, l'intelligence artificielle, la psychologie cognitive et la sociologie, s'adapte mieux à la réalité des systèmes complexes en réduisant la difficulté du processus de résolution de problèmes en le partageant entre plusieurs agents. Nous avons plus mis l'accent sur la coopération entre agents, car elle nous servira à mettre en place notre stratégie de sélection d'index et de vues matérialisées.

Deuxième partie

Contributions

Chapitre 4

Notre approche pour la sélection des index de jointure binaires

4.1 Introduction

Dans ce chapitre, nous présentons notre démarche de sélection automatique d'index dans les entrepôts de données schématisés en étoile. Cette démarche se base sur deux phases : (1) la sélection des index candidats et (2) la sélection d'une configuration finale d'index. La phase de sélection des index candidats nous permet de réduire l'espace de recherche en éliminant les attributs non pertinents.

Nous montrons en premier lieu les principales limitations des précédant travaux qui considèrent une approche data mining pour la sélection d'index. En se basant sur ces limitations, nous proposons une démarche distribuée à base d'un système multi agents coopératif pour l'extraction des itemsets fermés fréquents. Cette démarche utilise une nouvelle métrique d'élagage qui prend en compte un facteur pénalisant relatif à la qualité de l'index extrait et respecte en même temps le critère de monontonie et d'anti monotonie.

Enfin, nous proposons une autre stratégie utilisant un système multi agents et qui permet de sélectionner sous une contrainte de stockage, les index les plus avantageux grâce à un modèle de coût mathématique.

4.2 Limites des approches data mining pour la sélection d'index

Les deux travaux utilisant les techniques de data mining afin de réduire la complexité des algorithmes de sélection des index [12], [9] partent du principe que plus un attribut ou un groupe d'attributs est fréquemment présent dans la charge de requêtes plus il est intéressant de le(s) considérer dans le processus de sélection des index. Ensuite, des algorithmes de sélection sont exécutés pour générer les index finaux. Dans ce qui suit nous présentons les limites de ces approches et nous proposons par la suite des améliorations.

4.2.1 Limites de la fréquence comme paramètre d'élagage

La fréquence d'un itemset sur laquelle se base les approches [12] et [9] ne garantit pas à elle seule la qualité de l'itemset sélectionné.

Beaucoup de recherches ont montré la faiblesse des algorithmes qui utilisent uniquement le paramètre de fréquence comme métriques d'élagage [115], [139], [37].

Les travaux entrepris dans le cadre de la sélection d'un schéma de fragmentation verticale, ont aussi montré la limite des algorithmes de fragmentation basés sur la fréquence [87].

Par conséquent, nous affirmons que la fréquence d'un itemset ne garantit pas à elle seule la qualité d'un index, parce qu'elle ne contient pas d'informations sur l'avantage qu'il peut apporter [155], [157], [21], [22]. Pour confirmer ces limites dans le cadre de l'extraction des index de jointure binaires, nous considérons l'exemple suivant :

Exemple 1. Soit un ensemble de cinq requêtes définies sur un schéma en étoile, composé de deux tables de dimension *channels* et *customers* et une table des faits *sales*. La taille des tables (en termes d'instances) est :$\|Sales\| = 46521$, $\|Channels\| = 5$ et $\|Customers\| = 30000$.

(1) Select sales.channel_id, sum(sales.quantity_sold) sales.channel_id; from sales, channels where sales.channel_id=channels.channel_id and channels.channel_desc='Internet' group by sales.channel_id ;
(2) select sales.channel_id, sum(sales.quantity_sold), sum(sales.amount_sold) from sales, channels where sales.channel_id=channels.channel_id and channels.channel_desc ='Catalog' group by sales.channel_id ;
(3) select sales.channel_id, sum(sales.quantity_sold),sum(sales.amount_sold) from sales, channels where sales.channel_id=channels.channel_id and channels.channel_desc ='Partners' group by sales.channel_id ;
(4) Select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id=customers.cust_id and customers.cust_gender='M' group by sales.cust_id ;
(5) Select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id=customers.cust_id and customers.cust_gender='F' group by sales.cust_id ;

TABLE 4.1: Exemple de charge de requêtes

Supposons que *minsup*=3 (en valeur absolue). Un algorithme de sélection des index candidats en utilisant une approche basée sur l'extraction des itemsets fermés fréquents,

comme *Close* utilisée dans [7], retourne un index de jointure binaire construit sur la table *channels* et*Ssales* sur l'attribut *channels.channel_ desc* (figurant 3 fois dans les prédicats de sélection des requêtes).

L'index de jointure binaire entre la table de dimension *customers* et la table des faits *sales* (sur l'attribut *gender*) est élagué, car il n'est pas fréquent (il ne figure que deux fois). Mais ce dernier pourrait réduire le coût d'exécution de requête, car il est défini sur une table de dimension plus importante que la table *channels* (30 000 vs. 5).

Cet exemple montre l'insuffisance des travaux utilisant les approches d'extraction des itemsets fréquents ayant comme métrique d'élagage la fréquence d'apparition des attributs dans les requêtes. Nous affirmons que pour une bonne sélection d'index de jointures binaires, d'autres paramètres comme la taille des tables doivent être pris en considération.

4.2.2 Solution préconisée

L'exemple précédent a démontré les insuffisances des approches d'extraction des itemsets fréquents dans le contexte de sélection des index de jointure binaires. En conséquence, il faut les adapter afin de prendre en considération des paramètres autres que la fréquence d'accès, comme la taille des tables (la table des faits et les tables de dimension) concernées par l'indexation [156], [157], car le coût d'une opération de jointure dépend fortement de la taille des tables jointes [93].

Pour ce faire, nous proposons un nouveau paramètre, appelé, *fitness* qui sera utilisé comme une métrique d'élagage [157], [21]. La valeur de fitness est calculée dynamiquement à chaque itération, est défini par l'équation suivante :

$$fitness(X) = \frac{1}{n} \times (\sum_{i=1}^{n} sup_i \times \alpha_i) \tag{4.1}$$

où sup_i et n représentent respectivement le *support* et le nombre d'attributs non clés dans chaque itemset fermé fréquent X.

$\alpha_i$est égale à $\frac{|D_i|}{|F|}$ avec $|D_i|$et$|F|$ représentent respectivement le nombre de pages disque nécessaires à la table de dimension $D_i$et à la table de faits F.

Nous définissons aussi un seuil d'élagage minimum, appelé $MinFit$ défini par l'équation suivante :

$$MinFit = minsup \times |D_{min}| \tag{4.2}$$

où $minsup$ et $|D_{min}|$ représentent respectivement le *support* minimum et le nombre de page de la plus petite table de dimension.

4.2.3 Limite de la contrainte non anti monotone

La nouvelle contrainte $fitness$ a été établie en combinant la fréquence d'accès avec des paramètres relatifs à la taille des tables (la table des faits et les tables de dimension) concernées par l'indexation.

Cependant, [173] ont montré que toute les contraintes de la forme $avg(S)\theta v$, $median(S)\theta v$ et $sum(S)\theta v$ avec S pouvant contenir des éléments de valeurs arbitraires et $\theta \in \{>, <, \geq, \leq\}$ est v un nombre réel, ne sont ni monotone, ni anti-monotone.

Les travaux de [32], [92],[173], [193], [155], [156] ont aussi montré que la combinaison de la fréquence avec une contrainte non anti-monotone ne peut pas correctement élaguer l'espace de recherche.

Par conséquent, nous affirmons que la contrainte $fitness$ étant donné sa formulation ne garantit pas un bon élagage de l'espace de recherche. Pour confirmer cette limite dans le cadre de l'extraction des index de jointure binaires, nous considérons l'exemple suivant.

Exemple 2. Soit un ensemble de quatre requêtes définies sur un schéma en étoile composé de trois tables de dimension *Channels*, *Promotions* et *Customers* et une table des faits sales (Table 4.2). La taille des tables (en termes d'instances) est : $\|Sales\| = 1626033$ (la longueur d'une instance=36), $\|Customers\| = 700000$ (la longueur d'une instance =24), $\|Promotions\| = 18000$ (la longueur d'une instance =24), $\|Channels\| = 14000$(la longueur d'une instance =24). La taille d'une page disque PS=65536 octets. La cardinalité de *customers.cust_ gender*=2, *promotions.promo_ desc*=500, *channels.channel_ desc*=5.

(1) Select sales.channel_id, sum(sales.amount_sold) from Sales, Channels, Customers where sales.cust_id=customers.cust_id and sales.channel_id=channels.channel_id and channels.channel_desc= 'Catalog' and customers.cust_gender='F' group by sales.channel_id ;
(2) Select sales.channel_id, sum(sales.quantity_sold) from Sales, Channels, Customers where sales.cust_id=customers.cust_id and sales.channel_id=channels.channel_id and channels.channel_desc='Partners' and customers.cust_gender='M' group by sales.channel_id ;
(3) Select sales.channel_id, sum(sales.quantity_sold) from Sales, Channels where sales.channel_id=channels.channel_id and channels.channel_desc='Catalog' group by sales.channel_id ;
(4) Select sales.promo_id, avg(amount_sold) from Sales, Promotions where sales.promo_id=promotions.promo_id and promotions.promo_desc='internet' group by sales.promo_id ;

TABLE 4.2: Exemple de charge de requêtes

Pour faciliter la présentation de notre exemple, nous considérons des abréviations concernant les attributs extraits par l'étape de prétraitement des requêtes.

Abréviations	attributs
A1	*sales.cust_ id*
A2	*customers.cust_ id*
A3	*customers.cust_ gender*
A4	*sales.channel_ id*
A5	*channels.channel_ id*
A6	*channels.channel_ desc*
A7	*sales.promo_ id*
A8	*promotions.promo_ id*
A9	*promotions.promo_ desc*

TABLE 4.3: Liste des abréviations des attributs de notre exemple 2

Le tableau suivant donne les valeurs obtenues en utilisant l'algorithme *Close* ou *Dyna-close*.

1-Item	*support*	*Fitness*	*1-FCI*
A1	0.5	0.074804	A1A2A3A4A5A6
A2	0.5	0.074804	A1A2A3A4A5A6
A3	0.5	0.074804	A1A2A3A4A5A6
A4	0.75	0.005872	A4A5A6
A5	0.75	0.005872	A4A5A6
A6	0.75	0.005872	A4A5A6
A7	0.25	0.000839	A7A8A9
A8	0.25	0.000839	A7A8A9
A9	0.25	0.000839	A7A8A9

TABLE 4.4: Liste des itemsets fréquents fermés extraits

Pour *minsup* = 0.75, l'algorithme *Close* proposé dans [7] et [9], retourne un index de jointure binaire construit sur la table *Channels* et *Sales* sur l'attribut *Channels.channel_ desc* (A6) figurant 3 fois dans les prédicats de sélection des requêtes.

L'index de jointure binire entre la table de dimension *Customers* et la table des faits *Sales* sur l'attribut *Customers.cust_ gender*(A3) est élagué, car il n'est pas fréquent (il ne figure que deux fois). Cependant, ce dernier pourrait réduire le coût d'exécution des requêtes, car il est défini sur une table de dimension plus importante que la table *Channels* (700 000 30 000 vs. 14 000).

Supposons que *minsup*=1.00. Un algorithme de sélection des index candidats en utilisant une approche basée sur l'extraction des itemsets fermés, comme *Close* utilisée dans [7]et [9], ne retournera aucun index de jointure binaire, car tous les itemsets fermés sont

infréquents (voir Table 4.4). Dans ce cas, la charge de requêtes sera exécutée sans optimisation.

L'algorithme *Close* ne réduit pas correctement l'espace de recherche dans le contexte de la sélection des index, car la fréquence d'un itemset ne contient pas le profit qu'on peut tirer de son exploitation comme index. Ce résultat coïncide avec ceux des travaux menées par [87], [173].

Si on considère *minsup*=1.0 et on utilise *DynaClose*, on obtient les résultats suivants :

$$MinFit = \frac{\lceil \frac{14000\times 24}{65536} \rceil}{\lceil \frac{1626033\times 36}{65536} \rceil}$$

$$= 1\times \frac{6}{894}$$

$$= 0.\quad 006711$$

Les itemsets $A4; A5; A6; A7; A8; A9$ ne seront pas utilisés lors des itérations suivantes (voir tableau 4.4), parce que leurs fermés respectifs ont des valeurs de *Fitness* inférieures au seuil d'élagage $MinFit$.

Tous les 2-*itemsets* sont inclus dans les fermés fréquents générés précédemment et l'algorithme s'arrête. L'approche utilisée par *DynaClose*, conservera l'itemset fermé fréquent $FCI(X) = (A1A2A3A4A5A6)$ possédant les deux attributs indexables $A3; A6$, car la valeur de son *fitness* est égale à 0.074804 ($fitness(X) > MinFit$).

$$Fitness(X) = \frac{1}{2}\times \left[0.5 \times \frac{\lceil \frac{700000\times 24}{65536} \rceil}{\lceil \frac{1626033\times 36}{65536} \rceil} + 0.75 \times \frac{\lceil \frac{14000\times 24}{65536} \rceil}{\lceil \frac{1626033\times 36}{65536} \rceil}\right]$$

$$= \frac{1}{2} \times \left[0.5 \times \frac{257}{894} + 0.75 \times \frac{6}{894}\right]$$

$$= \frac{1}{2} \times [0.143736 + 0.05872]$$

$$= 0.0074804$$

DynaClose ne construit pas l'index généré à partir du sous-itemset $FCI(Y) = (A4A5A6)$ (voir tableau 4.4), car la valeur de son $Fitness$ est égale à 0.005872 ($fitness(Y) < MinFit$).

$$Fitness(Y) = 0.75 \times \frac{\lceil \frac{14000\times 24}{65536} \rceil}{\lceil \frac{1626033\times 36}{65536} \rceil}$$

$$= 0.75 \times \frac{6}{894}$$

$$= 0.005872$$

Cependant, partant de la propriété d'anti-monotonie des itemsets et qui stipule que tout itemset inclus dans un itemset fréquent est lui-même fréquemment, on devait avoir le sous-itemset $FCI(Y) = (A4A5A6)$ fréquent, car $FCI(Y) \subset FCI(X)$.

DynaClose ne peut pas correctement optimiser l'espace de recherche dans le problème de sélection des index de jointure binaires, car la métrique d'élagage *Fitness* ne respecte pas les relations anti-monotone : tout itemset inclus dans un itemset fréquent est lui-même fréquent. Notant aussi que la présence de cet index pouvait réduire le coût de traitement des requêtes en raison de la grande taille de la table *Products* (14 000 instances).

4.2.4 Solution préconisée

Dans le but d'améliorer l'efficacité de la métrique d'élagage, nous nous inspirons des travaux de recherche, qui se sont intéressés aux différentes façons de calculer et d'évaluer des conjonctions de contraintes anti-monotones et monotones [173], [32],[41],[29], [125], [61].

Nous nous basant notamment sur les travaux de [173], qui affirment que toute contrainte et toute combinaison de contrainte de la forme $avg(S)\theta v$, $median(S)\theta v$ et $sum(S)\theta v$, qui sont non anti-monotone et non monotone, sont trivialement convertibles en anti-monotone, convertibles monotone, seulement en choisissant un ordre sur les items.

4.3 Limites d'une approche de sélection centralisée

La grande volumétrie des données et la complexité des requêtes *OLAP* dans le contexte d'entrepôt de données, rend la sélection et la découverte des regroupements d'objets partageant des caractéristiques similaires trés complexe par une approche centralisée.

En effet si nous prenons le cas d'une base de données contenant n attributs booléen, le nombre total d'itemsets est égal à 2^n. Par exemple, pour un magasin avec 1000 d'articles mis en rayon et pour un support minimum $minsup$= 1 (en valeur absolue), le nombre de combinaison possibles pouvant être obtenu est proche de 10^{300}.

Plusieurs études [171], [230], [231], [196], ont montrées que l'approche d'extraction des itemsets fréquents est inapplicable aux données réelles. Ceci est dû au fait que le nombre d'itemsets fréquents généré est exponentiel par rapport au nombre d'attributs considérés.

Il faut rappeler que dans le contexte des entrepôts de données, le coût d'extraction de la représentation condensé des itemsets (les fermés fréquents) reste aussi très prohibitif malgré la réduction du nombre d'itemsets extraits.

4.3.1 Solution préconisée

Dans le but d'améliorer l'efficacité de notre approche de sélection, nous préconisons une démarche distribuée en utilisant un ensemble d'agents intelligents. Notre objectif est de diminuer la complexité de la sélection grâce à une distribution intelligente des données tout en diminuant le temps de sélection grâce à l'exploitation du parallélisme.

Nous préconisons de distribuer la charge de requêtes entre les différents utilisateurs de l'entrepôt. En effet, un même utilisateur suit en général un raisonnement logique dans son processus d'interrogation. De ce fait, les requêtes le concernant, le sont aussi. Cette corrélation entre ses requêtes donne lieu à un contexte d'extraction dense. Ceci permet d'améliorer notre approche d'extraction en temps de calcul et en espace mémoire nécessaires à la recherche des motifs fréquents fermés.

Nous avons aussi opté pour la génération distribuée des itemsets fermés qui constitueront les index candidats dans le processus d'indexation, car la représentation condensé des itemsets permet de produire un ensemble d'itemsets fréquents compact d'un point de vue taille et complet d'un point de vue connaissance [14], [23], [94], [196].

Enfin, plusieurs travaux [44], [45], [46], ont déjà montré que le data mining peut considérablement être amélioré par l'utilisation des agents, tandis qu'un système multi-agents peut bénéficier du data mining via l'extension des capacités des agents à découvrir la connaissance.

4.4 Les agents utilisés par notre approche

La principale particularité de notre approche par rapport à celles déjà existantes, est l'intégration d'une approche data mining à base d'agents intelligents. Pour mettre en œuvre notre approche, plusieurs types d'agents sont utilisés.

1) Agent coordinateur L'Agent COordinateur (*COA*) est doté d'un comportement cognitif. Son rôle est très important, car la phase d'élagage s'effectue à son niveau. L'objectif individuel d'un *COA* est de partager la charge de requêtes suivant les différentes sources d'interrogation. Pour cela, il interagit avec l'interface de l'utilisateur pour récupèrer la charge de requêtes et la valeur de *Minsup* que l'utilisateur introduit. Il a aussi comme objectif de générer les *FCIs* globaux à partir de l'ensemble des *FCIs* obtenues par les agents locaux. A cet effet il construit la représentation verticale globale des items grâce aux représentations partielles transmises par les différents agents locaux.

L'agent coordinateur utilise un moteur d'inférence qui exploite une base de règles et un ensemble de faits pour déduire de nouveaux faits et permettre à l'agent d'atteindre son but. Le tableau ci-dessous décrit certains faits utilisés par cet agent :

Fait	*Description*
F1	*MinSup*
F2	les informations relatives aux sources des requêtes d'interrogation
F3	les information contenues dans les Méta-data
F4	le *support* local obtenu par chaque *FCIs* extrait localement
F5	les *Tids relatifs à* chaque *FCIs* extrait localement
F6	les *FCIs* extrait par chaque agent local
F7	représentation verticale des items extraite par chaque agent local

TABLE 4.5: La base de faits de l'agent coordinateur

L'agent coordinateur utilise un ensemble de règles qui forment sa base de connaissance. Celles ci peuvent être décrite comme suit :

Règle	*Description*
R1	SI requête Q_i appartient à source interrogation S_i ALORS transmettre Q_i à l'agent local correspondant à S_i
R2	SI nouvelle source d'interrogation des requêtes trouvé ALORS créer son agent local
R3	SI $FCI(X_i) \cap FCI(X_j) \neq \emptyset$ ALORS garder $FCI(X) = FCI(X_i) \cap FCI(X_j)$ éliminer les fermés locaux$FCI(X_i)$ et $FCI(X_j)$ $Support(FCI(X)) = SupportFCI(X_i) + Support(FCI(X_j))$
R4	SI les fermés locaux $FCI(X_i) = FCI(X_j)$ ALORS garder $FCI(X_i)$ éliminer $FCI(X_j)$ $SupportFCI(X_i) = SupportFCI(X_i) + Support(FCI(X_j))$
R5	SI $Weight(FCI(X_j)) < MinWeight$ ALORS éliminer $FCI(X_j)$
R6	SI $Weight(FCI(X_j)) \geq MinWeight$ ALORS garder $FCI(X_j)$
R7	SI itemset $S = i_1, i_2, ..., i_m$ et $S' = i_1, i_2, ..., i_l$ avec $(l \leq m)$ ALORS $profit(S) \leq profit(S')$
R8	SI $profit(S) \leq profit(S')$ ALORS *ordre de parcours* SS'

TABLE 4.6: La base de règles de l'agent coordinateur

Le diagramme de cas d'utilisation ci-dessous montre les rôles d'un agent coordinateur.

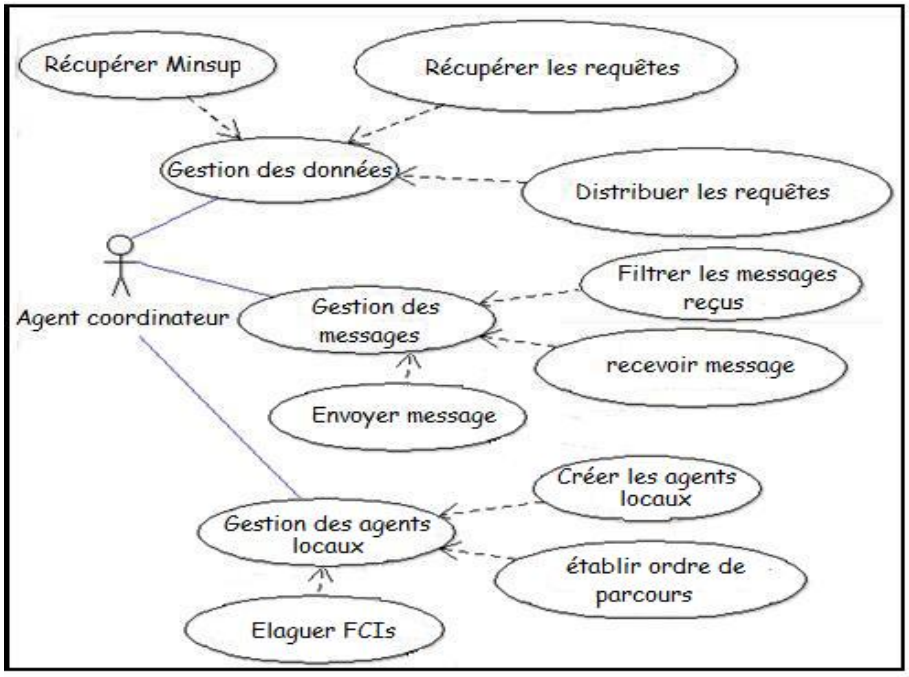

FIGURE 4.1: Diagramme de cas d'utilisation de l'agent coordinateur

2) Agent local On utilise un ensemble d'agents locaux (LA) qui sont dotés chacun d'un comportement réactif. Chaque agent local représente une source d'interrogation de l'entrepôt.

L'objectif individuel d'un *agent* local est l'extraction sans élagage des *FCIs* relatif à sa portion de requêtes. Cette extraction se fait indépendamment des autres agents. Le tableau ci-dessous décrit certains faits utilisés par cet agent :

Fait	*Description*
F1	Ensemble des requêtes transmise par le COA
F2	Ordre de parcours préconisé par le COA
F3	Ensemble des *FCIs* infréquents transmis par le COA

TABLE 4.7: La base de faits de l'agent local

L'agent local utilise un ensemble de règles qui forment sa base de connaissance. Le tableau ci-dessous donne un aperçu de certaines règles que l'agent local utilise.

Règle	*Description*
R1	SI $FCI(X)$ non validé par COA ALORS ne pas considérer $FCI(X)$ dans les prochaines itération
R2	SI $FCI(X_i)$ existe dans une requête Q_i ALORS $Support(FCI(X_i)) = Support(FCI(X_i)) + 1$ Mettre identifiant de la requête e Q_i avec les *Tids* du $FCI(X_i)$

TABLE 4.8: La base de de règles de l'agent local

Le diagramme de cas d'utilisation ci-dessous montre les rôles d'un agent local.

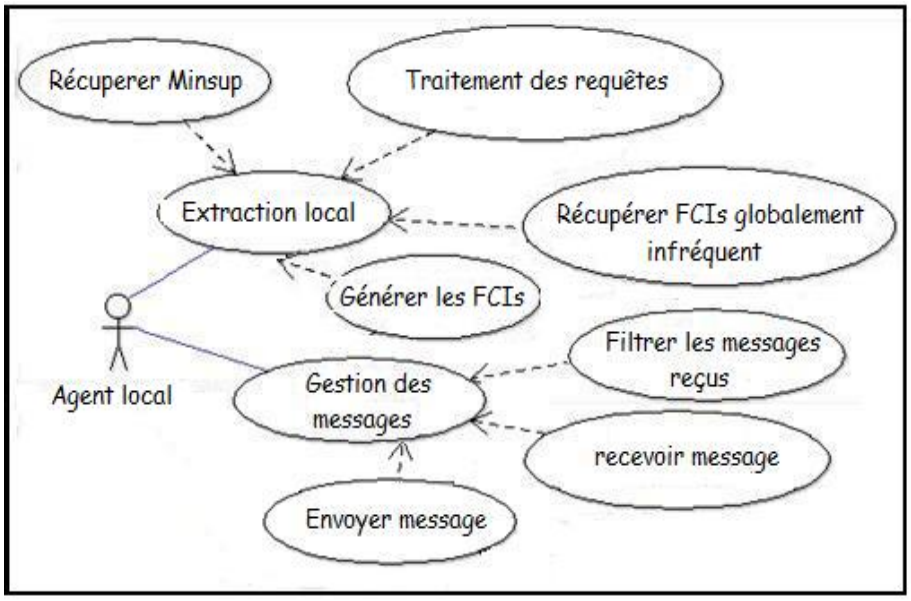

FIGURE 4.2: Diagramme de cas d'utilisation de l'agent local

3) Agent évaluateur L'agent évaluateur est doté d'un comportement cognitif. L'objectif individuel de cet agent est la sélection d'une configuration d'index de jointure binaires *(BJIs)* candidats. Il a aussi comme objectif l'extraction d'une configuration optimale de *BJIs* respectant la contrainte d'espace de stockage. Pour cela il utilise un modèle de coût mathématique. Le tableau ci-dessous donne un aperçu de certains faits que l'agent évaluateur utilise.

Fait	*Description*
F1	*Liste des FCIs valide*
F2	Liste et les identifiants de transaction (*Tid*) relatif à chaque *FCI*
F3	La représentation verticale globale des items
F4	La valeur *Weight* obtenue pour chaque *FCI*
F5	S l'espace de stockage alloué aux index
F6	Modéle de coût des index
F7	information sur les attributs formant les FCIs

TABLE 4.9: La base de faits de l'agent évaluateur

L'agent évaluateur utilise aussi un ensemble de règles qui forment sa base de connaissance. Le tableau ci-dessous donne un aperçu de certaines règles que l'agent évaluateur utilise.

Règle	*Description*
R1	SI $FCI(X)$ est défini sur N attributs de sélection de la table même dimension ALORS deux attributs clés sont nécessaires pour construire un BJI valide
R2	Si I $FCI(X)$ est défini sur k attributs de sélection de tables de dimension distinctes ALORS $2*k$ attributs clés sont nécessaires pour construire un BJI valide
R3	SI $S(BJI(X)) > S$ ALORS élimine $BJI(X)$
R4	SIBJI_j a un coût minimum et $S(BJI_j) \leq S$ ALORS BJI_j candidat pour la sélection finale

TABLE 4.10: La base de règles de l'agent évaluateur

Le diagramme de cas d'utilisation ci-dessous montre les rôles d'un agent évaluateur.

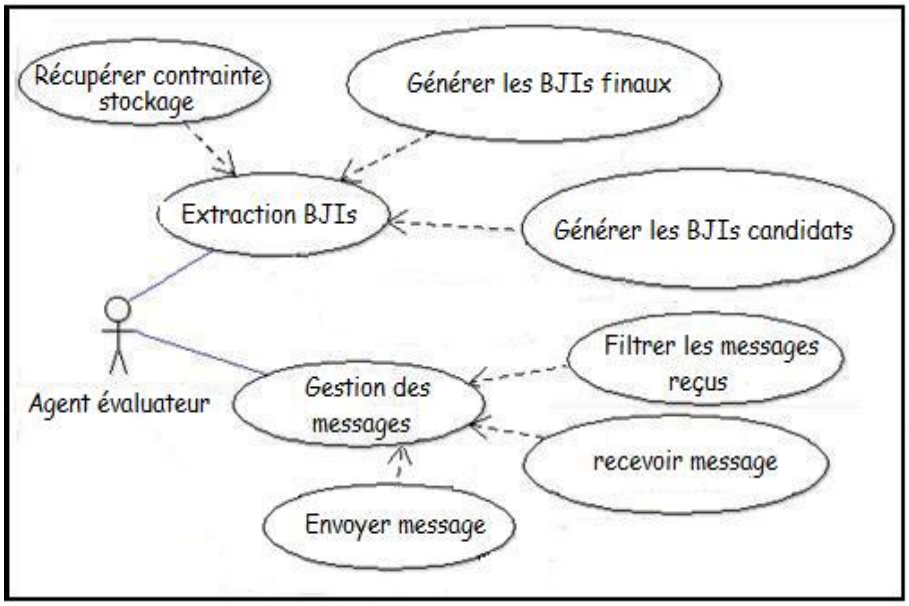

FIGURE 4.3: Diagramme de cas d'utilisation de l'agent évaluateur

4.5 Organisation de notre approche de sélection

Cette structure décrit comment les membres de notre système sont en relation afin d'atteindre un but commun en interagissant entre eux. L'organisation structurelle de notre système, est donnée par la figure suivante.

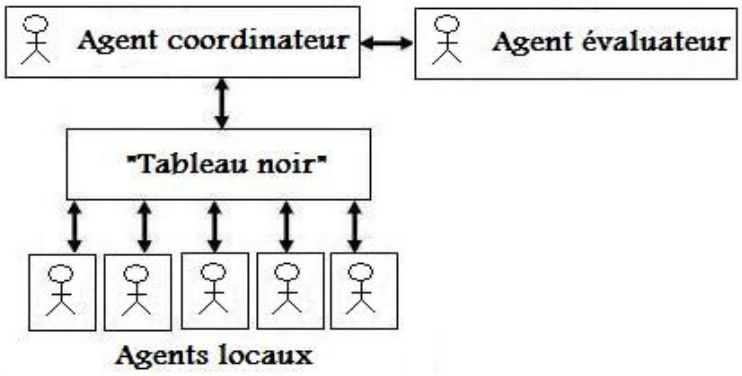

FIGURE 4.4: Architecture de notre approche de sélection

Nous remarquons qu'on a une architecture hiérarchisée composée essentiellement de deux niveaux. Nous n'avons aucune relation entre les agents locaux du même niveau. Cependant, on a des relations via le tableau noir entre les agents de niveaux différents.

Le modèle général de notre approche est aussi représenté par le diagramme de classes *AUML* : une classe générique (classe Agent) est la racine de la hiérarchie des classes représentant les agents modélisés.

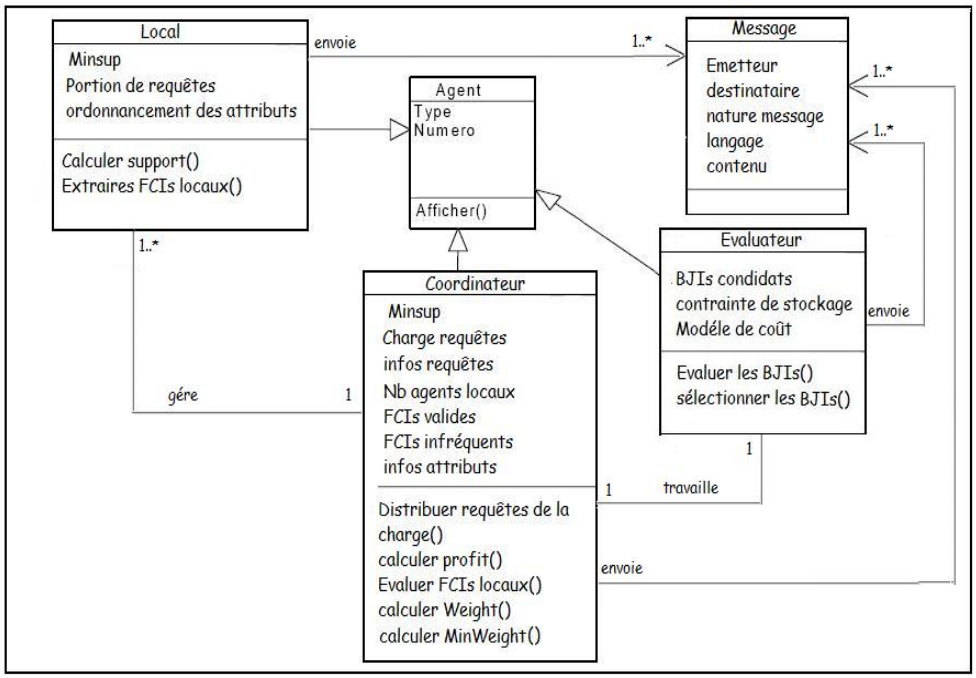

FIGURE 4.5: Diagramme de classe des agents de notre approche de sélection

4.6 Moyen et langage de communication de notre approche de sélection

Dans notre approche de sélection, nous avons adopté deux moyens de communication : le tableau noir et la communication par message. Le premier moyen a été choisi pour que

les agents locaux puissent communiquer avec l'agent coordinateur de façon indirecte afin d'optimiser la communication et le nombre de contrôles effectués. Le second moyen consiste à des échanges simples de type *FIPA-ACL* entre l'agent coordinateur et l'agent évaluateur.

4.7 Etapes de notre approche de sélection

Notre approche considère le problème de la sélection *BJIs* dans un environnement d'entrepôt de données modélisé en étoile, en utilisant six étapes principales : (1) Distribution des requêtes (2) Prétraitement locale des requêtes (3) Extraction locale des itemsets fermés (4) Etape d'élagage des itemsets fermés (5) la génération de *BJIs* et (6) la sélection d'une configuration finale de *BJIs*. Notant que la troisième et quatrième phase sont exécutées itérativement jusqu'à ce qu'une extraction d'itemsets fermés fréquents ne soit possible.

4.7.1 Distribution des requêtes

Cette étape commence lorsque le *COA* récupère la charge de requêtes à l'aide des fichier de journalisation et la valeur de *Minsup* que l'utilisateur introduit. Par la suite, il utilise les méta données de l'entrepôt et les informations relatives aux sources d'interrogation de l'entrepôt pour partager la charge de requêtes en portions qui seront gérées chacune par un des agents locaux qu'il crée. Cette distribution est faite de telle façon qu'une requête ne puisse être destinée qu'à un seul agent local.

Notant que cette étape est possible car les *SGBD* actuels et notamment *Oracle* [167] et *SQLServer* [39] gardent trace des requêtes d'interrogation et des informations relatives aux utilisateurs dans des fichiers de journalisation communément appelés *log*.

Le *COA* utilise également les méta données spécifiques qui décrivent l'information au sujet des requêtes, des tables et des attributs pour calculer le bénéfice de chaque attribut non-clé. Pour un attribut A_i cette valeur est calculée comme suit :

$$profit(A_i) = \frac{\left\lceil \frac{\|T_i\|}{C(A_i)} \right\rceil \times LT}{PS} \tag{4.3}$$

où $\|T_i\|$ et LT représentent respectivement le nombre d'instances, la longueur d'une instance de la table T. PS et $C(A_i)$ représentent respectivement la taille d'une page disque (en *octets*) et la cardinalité de A_i.

Le *COA* impose à chaque agent local d'utiliser un parcours par niveau dans l'ordre décroissant de *profit*. En effet, Pei et al., [173] ont démontré qu'un parcours dans le treillis des itemsets par ordre décroissant de $profit$, permet d'affirmer que le $profit$ d'un itemset $S = i_1, i_2, ..., i_m$ ne peut jamais être plus élevé que son préfix $S' = i_1, i_2, ..., i_l$ avec $(l \leq m)$. Ceci permet aussi de confirmer la monotonie et l'anti-monotonie de notre fonction *profit*.

Notant enfin que le *COA* récupère la valeur seuil *Minsup* que l'utilisateur introduit et la transmet aux différents agents locaux via le tableau noir. L'exemple suivant présente la proposition transmise par le COA sous forme d'un message Fipa-ACL.

(Propose

:sender (agent-identifier : COA)

:receiver (set(agent-identifier :CA))
:content "start mining with support=Minsup"
:reply with start mining proposal)

4.7.2 Prétraitement local des requêtes

Après réception de la valeur *Minsup* et des requêtes qui lui sont destinées, l'agent local extrait tous les attributs susceptibles d'être candidats pour l'indexation. Ces derniers sont ceux présents dans les clauses *WHERE* des requêtes.

Une fois l'extraction terminée, l'agent local construit une représentation verticale des items. Cette représentation matricielle consiste à mettre dans chaque ligne un attribut (item) extrait et les identifiants de transaction (*Tid*) qui le contiennent. A la fin de cette phase, chaque agent local transmet la représentation verticale de ces items au *COA*.

4.7.3 Extraction locale des itemsets fermés

Cette étape consiste à extraire localement les *FCIs* par un ensemble d'agents locaux. Chaque agent local procède de façon indépendante. Il n'y a pas de synchronisation lors de la recherche des itemsets fréquents fermés.

L'agent local utilise une stratégie inspirée de l'algorithme *Close* [170] pour extraire l'ensemble des itemsets fermés fréquents en utilisant un parcours par niveau dans l'ordre décroissant de *profit*.

Dans le but de rendre le processus l'extraction plus efficace, dans cette étape, seuls les attributs non-clés appelés attributs indexables sont utilisés dans le processus d'extraction. Rappelons qu'un *BJI* est défini entre la table de faits et une ou plusieurs tables de dimension sur les attributs indexables [167]. Notant aussi que cette étape qui se fait sans élagage, se termine par l'envoie au *COA* des *FCIs* extraits, leurs *Tids* et *support* respectifs.

4.7.4 Etape d'élagage des itemsets fermés

Cette étape permet à l'agent coordinateur de produire à partir des différentes solutions locales une sélection globale. Elle commence lorsque le CO*A* récupère de tous les agents locaux, l'ensemble des *FCIs* extraits, *Tids* et *supports* respectifs.

En premier lieu, le *COA* élimine les redondances des *FCIs* en additionnant les *supports* locaux d'un même *FCI* apparaissant plus d'une fois (voir ci-dessous la formule de calcul du *support* globale).

Par la suite, le *COA* réduit encore plus la liste des fermés, en établissant un ensemble d'intersection entre les différents *FCIs* extraits et en calculant les supports globaux. Ainsi on aura $FCI(X) = FCI(X_i) \cap FCI(X_j)$ pour toute paire d'itemsets fermés tel que $FCI(X_i) \subseteq FCI(X_j)$.

Le support globale d'un $FCI(X)$ est calculée (en valeur absolue) comme suit :

$$SupG(FCI(X)) = \sum_{i=1}^{n} (sup_i(FCI(X)) \tag{4.4}$$

où *supG* et sup_i représentent respectivement le *support* globale et le *support* local relatif à l'itemset fermé X.

Une fois la liste des itemsets fermés établi, le *COA* utilise une nouvelle métrique d'elagage pour déterminer les *FCIs* globalements fréquents. Cette métrique est calculée comme suit :

$$Weight(X) = \frac{supG(X)}{n} \times \lceil \sum_{i=1}^{n} (profit(A_i) \rceil \tag{4.5}$$

où *supG* et n représentent respectivement le *support* globale et le nombre d'attributs non clés dans chaque itemset fermé fréquent X .

$profit(A_i)$ représente le profit qu'on peut tirer de l'exploitation de l'item A_i comme index.

Le *COA* utilise un nouveau seuil d'élagage, appelé $MinWeight$ qui permet de pénaliser les *FCIs* ayant un petit *profit*. Ce seuil est calculé comme suit :

$$MinWeight = minsup \times Minprofit \tag{4.6}$$

où $Minprofit$ représente le $profit$ de valeur minimale.

Dans le but de réduire l'espace de recherche du problème de sélection d'index et à la différence des travaux existants, l*e COA* élague par rapport au seuil $MinWeight$ au niveau des fréquents fermés, car c'est eux qui formeront nos index candidats. A cet effet tout fréquent fermé dont la valeur $Weight$ est inférieure $MinWeight$ est considéré comme infréquent.

L'agent de coordination impose que seuls les préfixes dont les *FCIs* correspondants ont une valeur $Weight$ supérieure ou égale à $MinWeight$ seront utilisés pour générer le prochain ensemble de candidats ($Weight \geq MinWeight$).

Rappelons enfin que selon l'ordre d'extraction retenu (ordre descendant de *profit*), nous pouvons dire que la contrainte d'élagage $Weight$ respecte la propriété anti-monotonie, car chaque ajout au niveau des itemsets ne fera que diminuer la valeur du *profit*. Ainsi, les deux propriétés suivantes sont vérifiées :

a) Chaque fermé fréquent avec un préfixe non valide n'est pas valide.

b) Chaque préfixe d'un fréquent fermé qui satisfait la contrainte fréquent $Weight$ satisfait également cette contrainte.

[173] affirment que toute contrainte et toute combinaison de contrainte de la forme $avg(S)\theta v$, $median(S)\theta v$ et $sum(S)\theta v$, qui sont non anti-monotone et non monotone, sont trivialement convertibles en anti-monotone, convertibles monotone, juste en choisissant un ordre sur les items.

Notant enfin que cette phase qui est executée de façon itérative, vient à sa fin lorsque le *COA* enregistre tous les *FCIs* globalement fréquents et envoie l'ensemble des *FCIs* non fréquents à tous les agents locaux.

4.7.5 Construction des *BJIs* candidats

Cette phase débute lorsque l'agent évaluateur reçoit du *COA* l'ensemble des *FCIs* globalement fréquent. A cette fin l'agent évaluateur vérifie à partir de l'ensemble des *FCIS*

générés ceux qui ne respectent pas les caractéristiques de l'index de jointure binaires.

Pour réaliser cette tâche, l'agent évaluateur vérifie pour chaque *FCI* généré est-ce qu'il satisfait ou non les deux conditions suivante :

1. Si le *FCI* est défini sur des attributs de sélection d'une même table de dimension alors il doit exister une seule opération de jointure entre cette table et la table de faits. Ainsi, pour N attributs de sélection de la table même dimension, deux attributs clés sont nécessaires pour construire un *BJI* valide.
2. Si le *FCI* est défini sur k attributs de sélection de tables de dimension distinctes, il doit exister k opérations de jointures entre cette table et la table de faits. De ce fait on a *2*k* attributs clés.

Rappelant qu'un *BJI* valide composé de N attributs de sélection distincts doit contenir 2*n attributs clé qui représentent les prédicats de jointures [167].

Sur la base de ces exigences, l'agent évaluateur utilise via le tableau noir, la représentation verticale des items pour vérifier les attributs clés qui apparaissent toujours avec chaque *FCI* extrait.

Par la suite, il supprime tout *FCI* qui a un nombre d'attributs clés plus élevé ou plus petit que celui requit. Après cette opération, l'ensemble de *FCIs* purifiés est utilisé pour la construction d'un ensemble de *BJIs* candidats.

Pour illustrer le cas de rejet, prenons l'exemple suivant : considérons un itemset fréquent formé de (customers.cust_gender, sales.cust_id, customers.cust_id, sales.prod_id, products.prod_id) contenant trois attributs clés et un non clé. Si on veut construire l'index de jointure binaire, nous obtenons :

CREATE BITMAP INDEX sales_c_gender_p_cat_bjix

ON sales(customers.cust_gender)

FROM sales, customers, products

WHERE sales.cust_id = customers.cust_id *AND* sales.prod_id = products.prod_id

Cette instruction est fausse, car on a un attribut dans la clause *ON* et deux jointures dans le *WHERE*. Normalement, on doit avoir une seule jointure.

Exemple 3.

Dans ce qui suit, nous présentons l'exécution des étapes de notre approche.

Soit un ensemble de quatre requêtes définies sur un schéma en étoile composé de deux tables de dimension Channels, Promotions et Customers et une table des faits sales (Table 2). La taille des tables (en termes d'instances) est : $\|Sales\| = 1626033$ (la longueur d'une instance=36), $\|Customers\| = 700000$ (la longueur d'une instance =24), $\|Promotions\| = 18000$ (la longueur d'une instance =24), $\|Channels\| = 14000$(la longueur d'une instance =24). La taille d'une page disque PS=65536 octets. La cardinalité de *customers.cust_ gender*=2, *promotions.promo_ desc*=500, *channels.channel_ desc*=5.

(1) Select sales.channel_id, sum(sales.amount_sold) from sales, Channels, Customers where sales.cust_id=customers.cust_id and sales.channel_id=channels.channel_id and channels.channel_desc= 'Catalog' and customers.cust_gender='F' group by sales.channel_id ;
(2) Select sales.channel_id, sum(sales.quantity_sold) from sales, Channels, Customers where sales.cust_id=customers.cust_id and sales.channel_id=channels.channel_id and channels.channel_desc='Partners' and customers.cust_gender='M' group by sales.channel_id ;
(3) Select sales.channel_id, sum(sales.quantity_sold) from sales, channels where sales.channel_id=channels.channel_id and channels.channel_desc='Catalog' group by sales.channel_id ;
(4) Select sales.promo_id, avg(amount_sold) from sales, promotions where sales.promo_id=promotions.promo_id and promotions.promo_desc='internet' group by sales.promo_id ;

TABLE 4.11: Exemple de charge de requêtes

a) Prétraitement des requêtes

Cette phase nous permet d'extraire tous les attributs susceptibles d'être candidats pour l'indexation. Nous obtenons les mêmes attributs présentés dans l'exemple précédant, *à savoir :*

Abréviations	attributs
A1	*sales.cust_id*
A2	*customers.cust_id*
A3	*customers.cust_gender*
A4	*sales.channel_id*
A5	*channels.channel_id*
A6	*channels.channel_desc*
A7	*sales.promo_id*
A8	*promotions.promo_id*
A9	*promotions.promo_desc*

TABLE 4.12: Liste des abréviations des attributs de l'exemple 3

Etant donné que le calcul du *profit* se fait pour chaque attribut non clés extrait, nous obtenons les résultats suivants :

$$Profit(A3) = \frac{24 \times \lceil \frac{700000}{2} \rceil}{\lceil 65536 \rceil}$$

$$= 128.1738$$

$$Profit(A6) = \frac{24 \times \lceil \frac{18000}{500} \rceil}{\lceil 65536 \rceil}$$

$$= 0.0132$$

$$Profit(A9) = \frac{24 \times \lceil \frac{14000}{5} \rceil}{\lceil 65536 \rceil}$$

$$= 1.0254$$

b) Étape de distribution des requêtes

Nous considérons que dans notre cas, le *COA* partitionne l'ensemble des requêtes entre deux différents agents locaux (*LA*) qui représentent tous les départements de l'entreprise. Ainsi, le *COA* transmet les deux requêtes suivantes au premier agent local.

(1) Select sales.channel_id, sum(sales.amount_sold) from sales, Channels, Customers *where* sales.cust_id=customers.cust_id and sales.channel_id=channels.channel_id and channels.channel_desc= 'Catalog' and customers.cust_gender='F' group by sales.channel_id ;
(2) Select sales.channel_id, sum(sales.quantity_sold) from sales, Channels, Customers where sales.cust_id=customers.cust_id and sales.channel_id=channels.channel_id and channels.channel_desc='Partners' and customers.cust_gender='M' group by sales.channel_id ;

TABLE 4.13: Exemple de charge de requêtes pour l'agent local1

Le *COA* transmet les deux requêtes suivantes au deuxième agent local.

(3) Select sales.channel_id, sum(sales.quantity_sold) from sales, channels where sales.channel_id=channels.channel_id and channels.channel_desc='Catalog' group by sales.channel_id ;
(4) Select sales.promo_id, avg(amount_sold) from sales, promotions where sales.promo_id=promotions.promo_id and promotions.promo_desc='internet' group by sales.promo_id ;

TABLE 4.14: Exemple de charge de requêtes pour l'agent local2

Suivant les valeurs obtenues précédemment, le *COA* retient A3, A9, A6 comme ordre décroissant de *profit* et impose ceci aux deux agents locaux lors de l'étape d'extraction.

c) Étape d'extraction locale

A partir des deux requêtes transmises par le *COA*. L'agen local1 construit représentation verticale des items preséentée ci-dessous.

1-item	Tid
A1	1 2
A2	1 2
A3	1 2
A4	1 2
A5	1 2
A6	1 2

TABLE 4.15: représentation verticale des items obtenue par l'agent local1

Par la suite, l'agent local1 commence le processus d'extraction en utilisant seulement les attributs non-clés appelés attributs indexables. Le tableau suivant donne les valeurs obtenues par l'agent local1.

1-Item	*support*	*1-FCI*
A3	1.0	A3A6
A6	1.0	A6

TABLE 4.16: Liste des itemsets fréquents fermés extraits par l'agent local1

En même temps, et à partir des deux requêtes transmises par le *COA*, l'agent local2 construit la représentation verticale des items presentée ci-dessous.

1-item	Tid
A4	3
A5	3
A6	3
A7	4
A8	4
A9	4

TABLE 4.17: représentation verticale des items obtenue par l'agent local2

Par la suite, l'agent local2 commence le processus d'extraction en utilisant seulement les attributs non-clés. Le tableau suivant donne les valeurs obtenues par l'agent local2.

1-Item	*support*	*1-FCI*
A6	0.5	A6
A9	0.5	A9

TABLE 4.18: Liste des itemsets fréquents fermés extraits extraits par l'agent local 2

d) Étape d'élagage

Cette phase commence après que tous les agents locaux aient terminé leurs recherches et envoyé leurs résultats au *COA*. A partir des représentations locales obtenues par les deux agents locaux, le *COA* construit représentation verticale des items.

1-item	Tid
A1	1 2
A2	1 2
A3	1 2
A4	1 2 3
A5	1 2 3
A6	1 2 3
A7	4
A8	4
A9	4

TABLE 4.19: représentation verticale globale des items

Par la suite le *COA* calcule le *support* général des fréquents fermés globaux. On obtient les résultats suivants :

1-Item	*support*	*1-FCI*
A3	0.5	A3A6
A6	0.75	A6
A9	0.25	A9

TABLE 4.20: Liste des itemsets fréquents fermés extraits

Le *COA* calcule le seuil d'élagage minimal comme suit :

$$MinWeight = 1.0 \times 0.0132$$

$$= 0.0132$$

Dans le but d'évaluer la qualité de chaque *FCI* extrait, le *COA* calcule la métrique globale d'évaluation comme suit :

$$Weight(A3) = \frac{0.5}{2} \times \lceil 128.1738 + 0.0132 \rceil$$

$$= 64.0869$$

$$Weight(A6) = \frac{0.75}{1} \times \lceil 0.0132 \rceil$$

$$= 0.0099$$

$$Weight(A9) = \frac{0.25}{1} \times \lceil 1.0254 \rceil$$

$$= 0.2563$$

Suite aux résultats obtenues, le fréquent fermé $A6$ est élagué, car il a une valeur $Weight$ inférieure au seuil $MinWeight$.

Etant donné qu'après cette étape, les agents locaux ne peuvent extraire aucun 2-itemsets, l'algorithme s'arrête.

e) Construction des *BJIs* candidats

Dans le but d'éviter la construction des index erronés, l'agent évaluateur élimine l'ensemble des *FCIs* générés par la dernière étape et qui ne respectent pas les caractéristiques de l'index de jointure binaire. Dans cette étape, notre approche sélectionne :

- Le FCI1 = (A3) et donc suggère le *BJI* (appelé sales_gender_bjix) définie sur *costumers* et *sales* en utilisant l'attribut cust_gender. L'attribut customers.cust_gender qui n'a que deux valeurs distinctes (homme, femme), est optimal pour un index de jointure binaire.

La présence de cet index permet de réduire considérablement les besoins de stockage et améliore les performances des requêtes. Ce *BJI* optimise partiellement la requête (1) et (2).

- Le FCI2 = (A9) et donc suggère le *BJI* (appelé, sales_promo_desc_bjix) défini sur *promotions* et *sales* en utilisant l'attribut promotions.promo_desc. La présence de cet index améliore les performances de la requête (4). Cet index éviter complètement une opération de jointure pour la requête (4).

Nous voyons que sur trois *FCIS* extraits, nous avons deux *BJIs* valides. Notre approche fournit un meilleur rendement et améliore le temps de traitement (que la méthode traditionnelle). Elle respecte également le concept d'anti monotone et de monotonie (un itemset inclus dans un itemset fréquent est lui-même fréquent).

4.7.6 Modèle de coût d'exécution de requêtes en présence d'index

Dans le but d'estimer le coût des requêtes en présence ou non des index sélectionnés, nous avons utilisé un modèle de coût qui permet de calculer le nombre d'entrées sorties (E/S) nécessaires dans l'évaluation de chaque requête. Il permet aussi d'estimer la taille des index sélectionnés.

Notre modèle de coût possède les mêmes composantes généralement utilisées par la plus part des optimiseurs [100], [152], à savoir des statistiques sur les données et des formules pour évaluer les résultats des requêtes. Le tableau suivant présente l'ensemble de paramètres utilisés dans notre formulation.

Paramètres	*Description*
$\|\|T_i\|\|$	Nombre de nuplets de la table T_i
$\|T_i\|$	Nombre de pages disque de la table T_i
$w(T_i)$	Taille d'un attribut de la table T_i(en octets)
$Card(A_i)$	Cardinalité de l'attribut A_i
$max(A_i)$	valeur maximale de l'attribut A_i
$min(A_i)$	valeur minimale de l'attribut A_i
PS	Taille du pointeur de page (en octets)
π	Taille d'une page système
BJI	Index de jointure binaire

TABLE 4.21: Paramètres utilisés dans le modèle de coût

Etant donné que les tables relationnelles sont représentées par un ensemble de n-uplets partionnés en blocs liés par des pointeurs de blocs [20]. Dans cette étude, nous supposons que chaque n-uplet est assez petit pour tenir dans un bloc de données. Ceci a déjà été considéré dans les travaux de [20].

Le coût pour exécuter une requête depend notament du nombre d'accès au disque, de l'utilisation du *CPU* et de la mémoire [166]. Dans notre évaluation, nous considérons seulement le nombre d'accès au disque comme un facteur de coût à cause de son importance par rapport aux autres facteurs. Ainsi, le chemin d'accès détermine le nombre d'accès disque nécessaire pour l'obtention des données d'une table.

4.7.6.1 Evaluation d'une requête

Pour évaluer le coût d'une requête, nous utilisons la jointure par hachage. Nous avons opté pour cette méthode d'accès parce qu'elle est utilisée par plusieurs systèmes existants (e.g. *Oracle*, *Informix*).

La jointure par hachage se fait en deux phases [177]. Dans la première phase, les deux relations sont partitionnées suivants la même fonction de hachage appliquée aux attributs participant à la jointure. Dans la deuxième phase, les partitions en correspondance sont jointes [20].

Coût de la jointure Le coût d'une jointure est la combinaison des coûts d'accès individuels plus le coût de l'opération de jointure : boucles imbriquées, tri-fusion ou jointure par hachage.

Etant donné que nous avons opté pour la jointure par hachage, le coût de cette dernière entre deux tables en termes d'entrées sorties, est donné par [177] :

$$C(T_i, T_j) = 3 \times (|T_i| + |T_j|) \tag{4.7}$$

Coût de la sélectivité La sélectivité représente une partie d'un ensemble des lignes (tables, vue, résultat d'une jointure ou d'un opérateur GROUP BY) [166].

La sélectivité d'un prédicat indique combien de lignes d'un ensemble de lignes passent le test de filtrage du prédicat. Les valeurs de la sélectivité sont dans le domaine 0.0 à 1.0 où 0.0 signifie qu'aucune ligne ne sera sélectionnée et 1.0 indique que toutes les lignes seront sélectionnées.

Dans cette étude nous préconisons l'utilisation d'une démarche similaire à *Oracle9* [166] qui utilise la cardinalité comme filtre pour calculer le nombre des lignes ramenées. Cette démarche se base sur la théorie de probabilités qui nous enseigne que si nous jetons un dé et que nous nous posons la question de savoir combien de chances il y a d'obtenir la valeur 6. La réponse représente un divisé par le nombre des valeurs distinctes que le dé peut avoir ce qui vaut à 1/6.

D'une manière analogue, nous supposons que le nombre de lignes ramenées par une requête qui utilise un prédicat de type "col = 6" pour une table où il y a seulement six valeurs distinctes pour cette colonne, sera de 1/6 multiplié par le nombre des lignes de la table. Ainsi le coût de la sélectivité est calculé comme suit :

$$sel(A_i\text{=valeur}) = \frac{1}{card(A_i)}(\|T_i\|) \tag{4.8}$$

$$sel(A_i > valeur) = \frac{max(A_i) - valeur}{max(A_i) - min(A_i)} \tag{4.9}$$

$$sel(A_i < valeur) = \frac{valeur - max(A_i)}{max(A_i) - min(A_i)} \tag{4.10}$$

L'opération de sélection $sel(A_i \in (val1, val2))$ qui évalue le pourcentage de lignes pour lesquelles l'attribut A_i contient une valeur comprise entre on $val1$ et $val2$ est calculée comme suit :

$$sel(A_i \in (val1, val2)) = \frac{val1 - val2}{max(A_i) - min(A_i)} \tag{4.11}$$

4.7.6.2 Evaluation d'une requête en présence des index

L'évaluation d'une requête en présence des index de jointure binaires s'exécute selon les trois scénarii suivants :

Pas de correspondance Aucun index n'est pertinent pour la requête. L'évaluation du coût d'une requête consiste à additionner l'ensemble des coûts engendrés par les différentes opérations qu'elle contient.

Correspondance totale C'est le cas où un ou plusieurs index de jointure couvrent toutes les opérations de jointure de la requête. Pour estimer le coût de cette requête, nous utilisons un modèle de coût similaire à celui proposé par [7].

Le coût d'accès via un index dépend de nombre des niveaux d'index, de nombre des blocs feuille qu'il faut balayer et du nombre des lignes ramenées en utilisant le pointeur associé aux clefs d'index.

Dans ce cas, le coût d'exécution de la requête est divisé en plusieurs coûts correspondant aux trois étapes d'exécution et qui sont :

1. Le coût de descente du B-arbre de la racine jusqu'aux nœuds feuilles. Ce coût de descente du B-arbre vaut :

$$C_{descente} = log_m \|A\| - 1 \tag{4.12}$$

2. Le coût de parcours des nœuds feuilles afin de trouver les clés de recherche pour d bitmaps vaut :

$$C_{parcour} = \frac{\|A\|}{m-1} + d\frac{\|F\|}{8 \times PS} \tag{4.13}$$

3. Le coût de lecture des n-uplets et qui vaut :

$$C_{lecture} = |F| \times (1 - e^{-\frac{N_r}{PF}}) \tag{4.14}$$

Ainsi, le coût d'utilisation d'un index de jointure binaire pour évaluer une requête en utilisant un B-arbre pour accéder aux index vaut :

$$C_{index} = log_m \|A\| - 1 + \frac{\|A\|}{m-1} + d\frac{\|F\|}{8 \times PS} + |F| \times (1 - e^{-\frac{N_r}{PF}}) \tag{4.15}$$

où

$$m = \frac{PS}{w(A) + \pi} + 1 \tag{4.16}$$

La valeur de d est égale au nombre de prédicats appliqués sur l'attribut indexé et liés par l'opérateur *or* ou la cardinalité de la liste d'une clause *in*. Par exemple, la valeur de d de l'attribut indexé A est égale à 2 dans les deux clauses suivantes : A=5 *or* A=10 ; A *in* (5,10).

Ainsi le coût total d'exécution (CT) de l'ensemble des requêtes formant notre charge vaut :

$$CT = \sum_{i=1}^{n} f_i C(Q_i) \tag{4.17}$$

$C(Q_i)$ est le coût d'exécution de la requête.

Correspondance partielle Elle correspond au cas où l'index ne couvre que certaines tables. Il est alors utilisé pour traiter les tables qu'il couvre (comme pour la correspondance totale). Le résultat est combiné avec celui obtenu en effectuant des jointures classiques sur les tables non couvertes par l'index (comme dans le cas de non correspondance).

4.7.6.3 Coût de stockage d'un index de jointure binaire

Le stockage d'un index de jointure binaire dépend étroitement de la cardinalité de l'attribut indexé et du nombre de n-uplets de la table à laquelle il appartient. Les index de jointure binaires sont construits sur des attributs de tables dimensions et chaque bitmap

contient autant de bits que de n-uplets de la table de faits F. La taille de l'espace de stockage requis pour un index de jointure binaire construit sur un attribut A appartenant à une table T, est donnée par la formule suivante [226], [225] :

$$S(BJI(A)) = \frac{Card(A) \times |T|}{8} \tag{4.18}$$

4.7.6.4 Coût total d'exécution de l'ensemble des requêtes

Le coût total d'exécution de l'ensemble des requêtes est donné par :

$$CT = \sum_{i=1}^{m} f_i \times C(q_i) \tag{4.19}$$

Où $C(q_i)$ représente représentent respectivement le coût d'exécution de la requête q_i et sa fréquence d'apparition dans la charge.

4.7.7 Phase de génération de la configuration finale d'index

Rappelons que les itemsets fermés fréquents générés par l'étape précédente représentent des d'index candidats. Etant donnée la contrainte de stockage, ces derniers ne peuvent pas tous être utilisés pour l'indexation. Notre problème de sélection des index de jointure binaires est alors formalisé de la façon suivante :

- Etant donné un entrepôt de données modélisé par un schéma en étoile, formé d'une table des faits et des tables de dimensions.
- Un ensemble de requêtes les plus fréquentes avec leurs fréquences d'accès.
- Une configuration d'index de jointure binaires candidats.
- Une capacité de stockage S pour stocker les index sélectionnés.

L'objectif est de sélectionner un ensemble d'index réduisant le coût d'exécution de requêtes et satisfaisant la contrainte S.

Pour réaliser cette tâche, l'agent évaluateur utilise le modèle de coût mathématique précédemment présenté afin d'évaluer chaque BJI candidat.

L'agent évaluateur procède en deux étapes. Dans la première, il élimine tous les index de jointure binaires qui ne respectent pas la contrainte d'espace disque S.

Dans la seconde étape, il sélectionne une configuration initiale qui comprend l'index réduisant le plus le coût d'exécution des requêtes. Ensuite, il enrichit d'une manière itérative cette configuration en considérant d'autres index, tant que le coût de stockage n'est pas dépassé. Les principales étapes de cette approche sont décrites dans Algorithme ci-dessous.

Algorithm 4.1 Algorithme de sélection finale des index

Entrée : $ConfigCand$: ensemble de BJIs candidats,

S : contrainte de stockage,

Sortie : $ConfigF$: ensemble final de BJIs,

DEBUT

Evaluer le coût de chaque $BJI_j \in ConfigCand$

Tant que $(ConfigCand \neq \varnothing)$et $(S(BJI_j) > S)$ faire

$ConfigCand = ConfigCand - (BJI_j)$

fait ;

$ConfigF$ =Coût des requêtes sans indexation

$S(ConfigF) = 0$

Tant que $(ConfigCand \neq \varnothing)$et $(S(ConfigF) \leq S)$ faire

Si $C(BJI_j) = Min(C(BJI_j), Q_i)$ et $C(ConfigF \cup BJI_{min}) < C(ConfigF)$ alors

$ConfigF = ConfigF \cup BJI_{min}$

$S(ConfigF) = S(BJI_{min})$

$ConfigCand = ConfigCand - (BJI_{min})$

finsi

Fait

FIN. // Ensemble final de BJIs

4.8 Expérimentations

Notre étude expérimentale utilise un schéma en étoile, généré par le benchmark *APB-1* [161], composé d'une table des faits *Activars* et de quatre tables de dimension : *Prodlevel*, *Custlevel*, *Chanlevel*, *Timelevel*. Le tableau suivant présente les informations concernant les tables composant notre entrepôt de données.

Paramètres	*Valeur*
$\|\|Activars\|\|$	24 786 000
$w(Activars)$	74
$\|\|Prodlevel\|\|$	10 000
$w(Prodlevel)$	72
$\|\|Custlevel\|\|$	1000
$w(Custlevel)$	24
$\|\|Timelevel\|\|$	24
$w(Timelevel)$	36
$\|\|Chanlevel\|\|$	9
$w(Chanlevel)$	24
PS	65536
π	4096

TABLE 4.22: Valeurs des tables utilisées dans notre évaluation de l'approche de sélection d'index

Le tableau ci-dessous présente les cardinalités concernant les attributs de sélection des tables de dimension utilisés dans nos évaluations expérimentales.

Table	attribut	Cardinalité
ProdLevel	Class_level	605
	Group_level	300
	Family_level	75
	Line_level	15
	Division_level	4
TimeLevel	Year_level	2
	Month_level	12
CustLevel	Retailer_level	99
ChanLevel	All_level	9

TABLE 4.23: Cardinalités des attributs de sélection utilisés dans nos évaluations de l'approche de sélection des index

Pour valider notre approche de sélection, nous utilisons *JADE* (Java Development Framework Agent) comme outil pour mettre en œuvre notre infrastructure. Nous avons utilisé aussi une charge formée de 310 requêtes décisionnelles et six agents locaux car il nous était impossible d'extraire les requêtes sans l'utilisation d'une application payante comme Lumigent's Log Explorer [146].

Nos expérimentations se sont déroulées selon trois scénarios : (1) l'identification des valeurs de *Minsup* qui permettent de donner des itemsets fréquents, (2) l'évaluation de notre démarche sans considération pour l'espace de stockage, en testant les 310 requêtes

décisionnelles sur l'entrepôt non indexé, indexé avec notre approche et (3) l'évaluation de notre approche avec la contrainte d'espace de stockage et (3) mesure du temps *CPU* lié à l'exécution de notre approche.

4.8.1 Evaluation sans contrainte d'espace

La Figure 4.6 montre comment les différentes approches d'indexation améliorent le temps d'exécution des 310 requêtes de la charge. Avec un *MinWeight*=0,0000106319, nous avons obtenu un gain de 47,59 % pour notre stratégie sur la stratégie non indexée. Ce taux parait très important étant donné le nombre élevé d'itemsets fermé extrait et de ce fait le nombre important d'index de jointure à considérer.

Le principal résultat est que notre approche offre les meilleures performances pour la plus parts des valeurs de *MinWeight,* mais ne dégrade jamais ces performances. Notons aussi que les performances de toutes les approches se détériorent pour des valeurs élevées de *MinWeight* qui réduisent le nombre de *BJIs* générés.

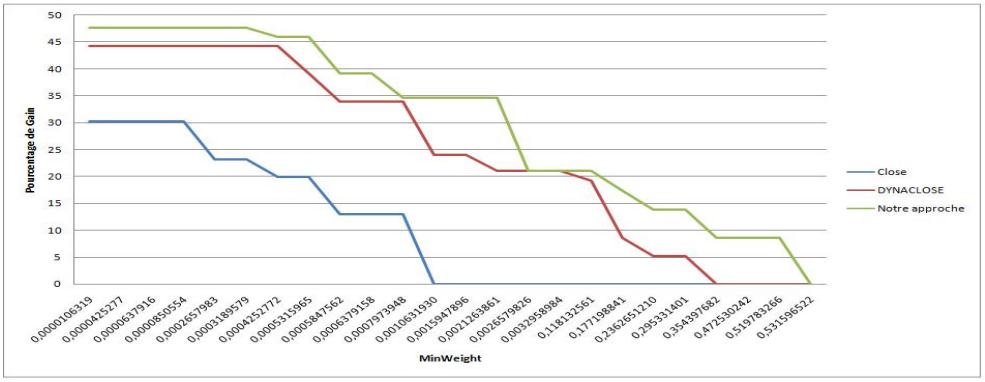

FIGURE 4.6: Qualité de notre approche de sélection des index

Pour des valeurs *MinWeight* supérieures à 0,0007973948, *Close* cesse de générer de nouveaux itemsets fermés et le coût du traitement des requêtes reste stable.

Pour une valeur *MinWeight* supérieure à 0,295331401 *DynaClose* cesse de produire des index et son coût devient égal à celui sans indexation au moment ou notre approche génère toujours des index jusqu'à atteindre une valeur *MinWeight* supérieure à 0,519783266.

4.8.2 Evaluation avec contrainte d'espace

Notons que l'ensemble des index de jointure candidats, généré par notre approche ne peut pas être complètement sélectionnés dans la pratique, compte tenu de l'espace de stockage requit et qui dépasse les 34oo Mo.

En conséquence, nous avons exécuté notre approche de sélection en tenant compte des différentes valeurs de stockage pour une valeur fixe de *MinWeight* égale à 0,0004252772. Cette valeur permet la génération d'un grand nombre d'index candidats.

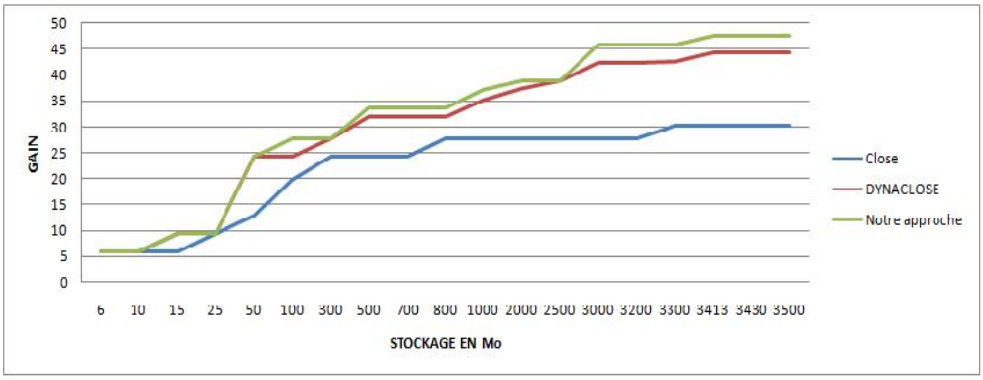

FIGURE 4.7: Evaluation avec contrainte d'espace de notre approche de sélection des index

La figure 4.7 montre que notre approche fournit une meilleure performance que les approches précédentes pour la plupart des espaces de stockage considérés.

Pour l'ensemble des approches testées, la figure 4.7 montre aussi que plus on dispose d'espace, plus on améliore le temps d'exécution des requêtes. Ceci parait évident, étant donné que l'existence de plus d'espace de stockage permet la sélection de plus d'index.

Nous remarquons aussi que pour les valeurs de stockage dépassant 3200 Mo, Close *cesse d'améliorer notre charge de requêtes.* Notre approche et *DynaClose* cessent d'améliorer notre charge de requêtes pour les valeurs de stockage dépassant 3400 Mo.

Pour conclure, notre approche nous permet de faire un meilleur choix entre les différents index candidats. Ceci permet d'avoir un taux d'amélioration qui demeure appréciable pour tous les espaces de stockage pris en considération.

4.8.3 Evaluation temps d'exécution

Nous avons mené un ensemble d'évaluations expérimentales sur le temps de traitement de chaque approche (en millisecondes) conformément à différentes valeurs de *MinWeight.*

La figure 4.8 montre que l'algorithme *Close* nécessite le moins de temps pour les grandes valeurs *MinWeight,* car cet algorithme élague les itemsets fermés selon la valeur *Minsup* seulement. Ceci est prévisible, car pour des valeurs élevées de *MinWeight* le nombre d'itemsets fermés est très réduit.

Nous remarquons aussi que *Dynaclose* est l'approche qui exige le plus grand laps de temps pour la quasi-totalité des valeurs de *MinWeight.* Ce résultat était prévisible puisque *Dynaclose* utilise tous les attributs et élague grâce à une contrainte non anti-monotone. Ces résultats coïncident avec l'étude expérimentale de Bellatreche et al., [21].

Notre approche exige des temps d'exécution très petits pour presque toutes les valeurs de *MinWeight.* Ceci était prévisible, car notre approche se base sur une démarche distribuée qui utilise uniquement les attributs non-clés dans la phase d'extraction des index candidats.

Le temps de traitement nécessaire pour *Close*, *Dynaclose* et notre approche diminue pour les grandes valeurs de *MinWeight.* Ces résultats sont dû au fait que les grandes valeurs de *MinWeight* diminuent le nombre des index extraits.

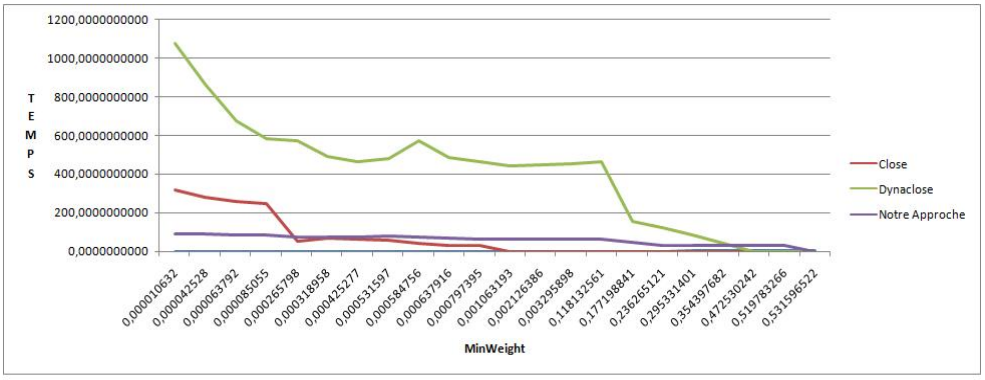

FIGURE 4.8: Evaluation temps d'exécution de notre approche de sélection des index

4.9 Conclusion

Dans ce chapitre, nous avons présenté notre démarche de sélection automatique d'index de jointure binaire pour les entrepôts de données schématisé en étoile. Cette démarche se base sur deux phases : (1) la sélection des index candidats et (2) la sélection d'une configuration finale d'index. La phase de sélection des index candidats nous permet de réduire l'espace de recherche en éliminant les attributs non pertinents.

Pour élaguer l'espace de recherche des index, nous avons en premier lieu montré les principales limitations des précédant travaux qui considèrent seulement la fréquence d'accès des requêtes ou une contrainte non anti-monotone dans le processus de génération des motifs fréquents.

En se basant sur ces constats, nous avons proposé une démarche de sélection distribuée à base d'un système multi agents coopératif. Notre objectif est de diminuer la complexité de la sélection grâce à une distribution intelligente des données tout en diminuant le temps de sélection grâce à l'exploitation du parallélisme.

Nous avons opté pour la génération distribuée des itemsets fermés qui constitueront les index candidats dans le processus d'indexation, car la représentation condensé des itemsets permet de produire un ensemble d'itemsets fréquents compact d'un point de vue taille et complet d'un point de vue connaissance

Nous avons aussi utilisé une nouvelle métrique d'élagage qui prend en compte un facteur pénalisant relatif au nombre de pages et à la cardinalité des différentes tables de dimension par rapport à la table des faits. Ce choix repose sur le fait que toute opération de jointure depend fortement de la taille des tables jointes et les index de jointure binaires sont recommandés pour des attributs ayant une faible cardinalité. Dans le but d'optimiser correctement l'espace de recherche, notre métrique d'élagage respecte aussi les relations de monotonie et d'anti-monotone.

Enfin,pour la phase de sélection d'une configuration finale d'index, nous avons aussi proposé une autre stratégie utilisant un système multi agents permettant de sélectionner sous une contrainte de stockage, les index les plus avantageux grâce à un modèle de coût mathématique.

Nos résultats expérimentaux montrent que notre stratégie de sélection d'index reste la meilleure approche pour améliorer les performances. Cette amélioration des coûts demeure appréciable pour tous les espaces de stockage pris en considération.

Chapitre 5

Notre approche pour la sélection des vues matérialisées

5.1 Introduction

La matérialisation des vues est l'une des approches les plus couramment utilisée pour optimiser les performances pour les requêtes de type *OLAP* connues pour leur grande complexité.

Dans ce chapitre, nous présentons notre approche de sélection d'un ensemble optimal de vues à matérialisées. Notre démarche utilise la technique de clustering distribuée à base d'agents coopératifs.

Notre objectif est de profiter de la coopération d'un ensemble d'agents pour réduire la complexité du processus de sélection d'un ensemble des vues matérialisées. L'ensemble des vues sélectionnées, permet de minimiser au mieux le coût de traitement des requêtes et satisfait en même temps une contrainte de stockage.

5.2 Motivations de notre approche

Dans le contexte *OLAP*, les performances des requêtes d'interrogation peuvent être grandement améliorées en stockant des résultats intermédiaires dans les vues matérialisées [204], [181], [1], [13], [116], [228], [99],[129].

Cette amélioration vient du fait que certaines requêtes peuvent être traitées efficacement, car elles nécessitent seulement l'accès à ces vues matérialisées sans avoir besoin d'accéder aux données d'origine [159]. De plus, nous avons également la possibilité d'indexer ces vues, ce qui améliore encore plus leur efficacité.

Dans notre cas, le schéma en étoile est considéré comme une vue conceptuelle sur laquelle des vues matérialisées peuvent être introduites pour accélérer les traitements.

L'idée de base de notre approche est de regrouper par clustering les parties des requêtes syntaxiquement proches puisqu'elles ont une forte probabilité d'être résolues par une simple vue matérialisée [158].

Toutefois, la méthode de clustering nécessite un temps de calcul très élevé [149], ce qui limite son utilisation à des données de taille relativement restreinte.

Le volume de données et de la complexité des requêtes *OLAP* dans le contexte des entrepôts de données modélisé en étoile, nous incitent à adopter une approche d'intelligence distribuée [30] et d'utiliser les avantages de l'approche multi-agents tels que : le parallélisme, la robustesse [210] pour améliorer les processus de sélection et réduire la complexité de ce problème [158].

Nous préconisons de distribuer la charge de requêtes entre les différents utilisateurs de l'entrepôt. En effet, un même utilisateur suit en général un raisonnement logique dans son processus d'interrogation. De ce fait, les requêtes le concernant, le sont aussi. Cette corrélation entre ses requêtes donne lieu à un contexte d'extraction dense. Ceci permet d'améliorer notre approche d'extraction en temps de calcul et en espace mémoire nécessaires à notre démarche de classification non supervisée.

Nous adaptons l'algorithme de classification non supervisée, *K-means* [148] qu'on a mentionné dans l'état de l'art, afin de pallier aux inconvénients du *k-means* classique [148] et notamment le paramètre K qui représente le nombre initial de classe que l'utilisateur doit fixer au préalable. Ce paramètre s'avère empirique et imprécis, car le nombre optimal de classes ne peut être trouvé qu'après plusieurs expérimentations.

Pour remédier à ce problème, notre algorithme introduit la notion d'incrémentation. En effet, notre approche ne nous oblige pas à déterminer un nombre initial de classe au début. Mais au fur et à mesure que nous classons nos données, nous créons de nouvelles classes pour les prédicats n'ayant pas été acceptés dans les classes déjà existantes [158].

5.3 Les agents utilisés par notre approche

La principale particularité de notre approche par rapport à celles déjà existantes, est l'intégration d'une approche data mining à base d'agents coopératifs. Pour mettre en œuvre notre système de sélection, plusieurs types d'agents sont utilisés.

1) Agent manager L'agent manager est doté d'un comportement cognitif. Son objectif individuel est de partager la charge de requêtes suivant les sources d'interrogation. Il crée à cet effet un ensemble d'agents coordinateurs. Pour cela il a les compétences suivantes :

- L'interaction avec l'interface de l'utilisateur en récupérant la valeur de *MinDist*,
- la création des agents coordinateurs,
- la gestion des messages,

La base de faits de l'agent manager contient entre autres les faits suivants :

Fait	*Description*
F1	*MinDist*
F2	les informations relatives aux sources des requêtes d'interrogation
F3	les informations contenues dans les Méta-data

TABLE 5.1: La base de faits de l'agent manager

Le tableau ci-dessous donne un aperçu de certaines règles que l'agent manager utilise :

Règle	*Description*
R1	SI requête Q_i appartient à source interrogation S_i ALORS transmettre Q_i à l'agent coordinateur correspondant à S_i
R2	SI nouvelle source d'interrogation des requêtes trouvé ALORS créer son agent coordinateur

TABLE 5.2: La base de règles de l'agent manager

Le diagramme de cas d'utilisation ci-dessous montre les rôles de l'agent manager.

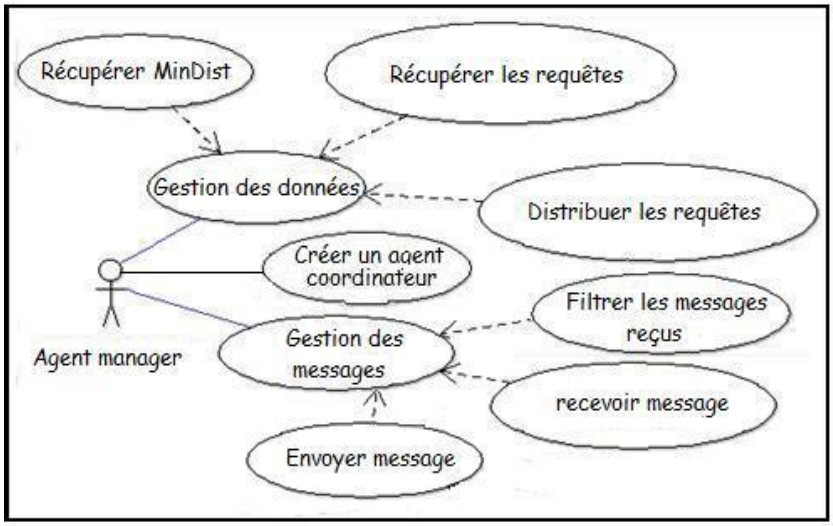

FIGURE 5.1: Diagramme de cas d'utilisation de l'agent manager

3) Agent coordinateur On utilise un ensemble d'agents coordinateurs. Chaque agent coordinateur (COA) est doté d'un comportement cognitif et représente une source d'interrogation de l'entrepôt.

L'objectif individuel d'un agent coordinateur est la bonne gestion des agents clusters qu'il crée. Il coordonne aussi avec les autres $COAs$ afin de permettre la convergence de toutes les solutions locales. Le tableau ci-dessous donne un aperçu de certains faits que l'agent coordinateur utilise.

Fait	*Description*
F1	*MinDist*
F2	délai de réponse pour chaque proposition
F3	Liste des requêtes transmises par l'agent manager

TABLE 5.3: La base de faits de l'agent coordinateur

L'agent coordinateur utilise l'ensemble de règles afin d'atteindre ces objectifs. Le tableau ci-dessous donne un aperçu de certaines règles que l'agent coordinateur utilise.

Règle	*Description*
R1	SI requêteQ_i utilise prédicat P_i ALORS Mettre 1 dans la cellule de la matrice d'usage
R2	SI requêteQ_i n'utilise pas le prédicat P_i ALORS Mettre O dans la cellule de la matrice d'usage
R3	SI $Dist(o_i, c_i)$ a la plus petite valeur ALORS assigner le prédicat à l'agent cluster offrant $Dist(o_i, c_i)$
R4	SI aucune offre intéressante n'existe ALORS créer un nouvel agent cluster et lui assigne le prédicat
R5	SI aucune offre d'échange n'a été enregistrée ALORS rejeter l'échange
R6	SI plusieurs offres d'échange reçues ALORS choisir meilleur offre
R7	SI (aucune offre d'échange n'est interessante) ET (évaluation du cluster sans la présence du prédicat est meilleur) ALORS Céder le prédicat sans contre partie
R8	SI($Dist(p_i, c_j) \leq MinDist$) ET ($p_i$ améliore moyenne cluster en comptabilisant p_i) ALORS accepter proposition d'échange
R9	SI$Dist(p_i, c_j) > MinDist$ ALORS refuser le prédicat p_j $MinDist$
R10	SI prédicat p_i existe dans cluster ALORS ne pas participer aux offres d'échange concernant le prédicat p_i
R11	SI délai offert écoulé ALORS mettre refus pour tout agent n'ayant pas répondu

TABLE 5.4: La base de règles de l'agent coordinateur

Le diagramme de cas d'utilisation ci-dessous montre les rôles de l'agent coordinateur.

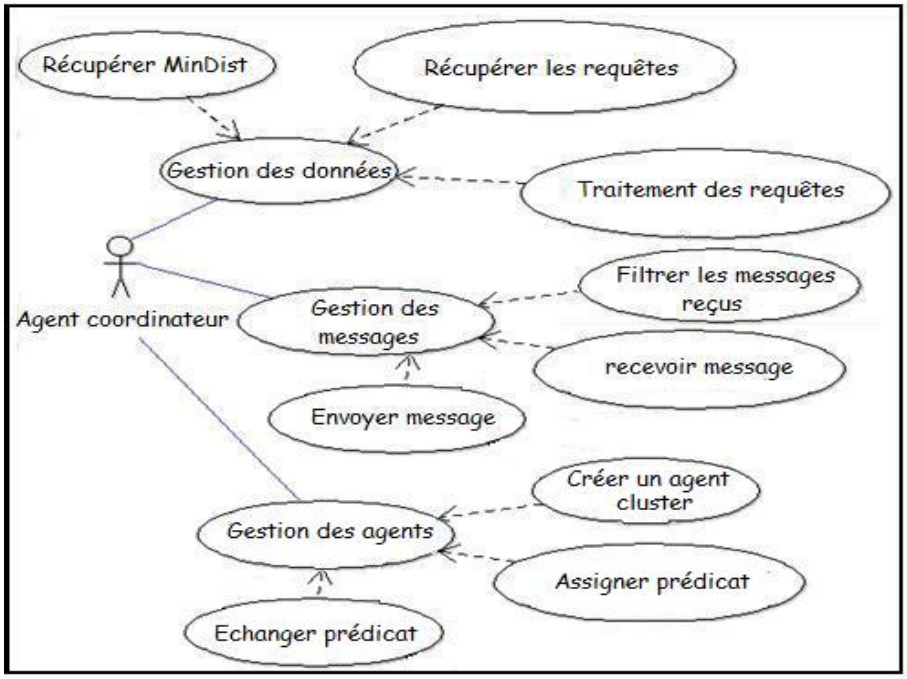

FIGURE 5.2: Diagramme de cas d'utilisation de l'agent coordinateur

4) Agent Cluster On utilise un ensemble d'agents cluster (*CA*). Chaque agent cluster est doté d'un comportement réactif. Il est sollicité par le *COA* pour gérer un ensemble de prédicats qui sont très similaires. Le tableau ci-dessous donne un aperçu de certains faits que l'agent Cluster utilise.

Fait	*Description*
F1	*MinDist*
F2	La matrice de contexte
F3	délai de réponse transmis par le *COA*

TABLE 5.5: La base de faits de l'agent cluster

L'agent cluster utilise un ensemble de règles simples afin d'accomplir correctement sa tâche. Le tableau ci-dessous donne un aperçu de certaines règles que l'agent cluster utilise.

Règle	*Description*
R1	SI p_i premier prédicat reçu ALORS considérer le prédicat p_j comme centre de gravité
R2	SI $Dist(p_i, c_j) > MinDist$ ALORS refuser le prédicat p_i
R3	SI($Dist(p_i, c_j) \leq MinDist$) ET ($p_i$ améliore le cluster en comptabilisant p_i) ALORS accepter proposition
R4	SI prédicat p_i existe dans cluster ALORS ne pas participer aux offres concernant le prédicat p_i
R5	SI aucune proposition n'a été enregistrée ALORS annuler offre
R6	SI plusieurs propositions reçues ALORS choisir meilleur offre

TABLE 5.6: La base de règles de l'agent cluster

Le diagramme de cas d'utilisation ci-dessous montre les rôles d'un agent cluster.

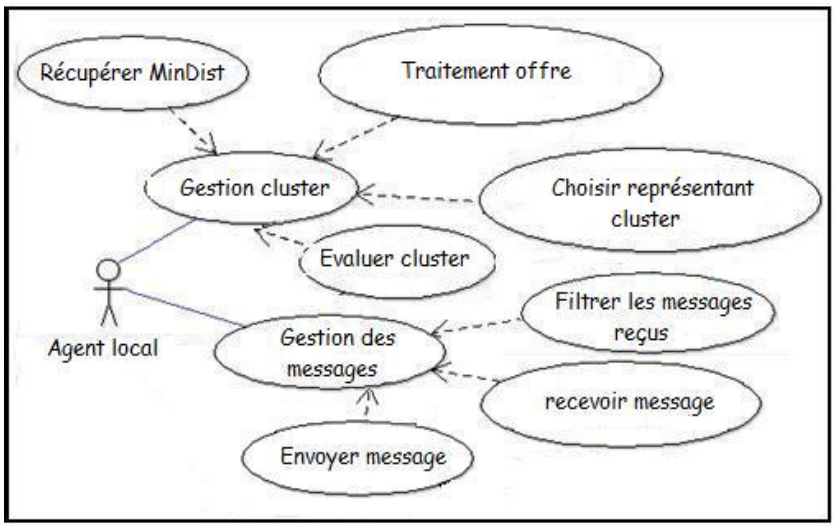

FIGURE 5.3: Diagramme de cas d'utilisation de l'agent cluster

5) Agent fusion L'agent fusion est doté d'un comportement cognitif. L'objectif individuel d'un agent fusion est la sélection d'une configuration de vues candidates. Il a aussi comme objectif l'extraction d'une configuration optimale de vues matérialisées respectant la contrainte d'espace de stockage. Pour cela il utilise un modèle de coût mathématique. Le tableau ci-dessous donne un aperçu de certains faits que l'agent fusion utilise.

Fait	*Description*
F1	S l'espace de stockage allouer aux vues
F2	Modéle de coût des vues
F3	Ensemble des clusers

TABLE 5.7: La base de faits de l'agent fusion

L'agent fusion utilise un ensemble de règles formant sa base de connaissances. Le tableau ci-dessous donne un aperçu de certaines règles que l'agent fusion utilise.

Règle	*Description*
R1	SI $(C(V) \leq C(V_i) + C(V_j))$ et $(S(V) \leq S(V_i) + S(V_j))$ ALORS considérer V et éliminer etV_i et V_j
R2	SI $S(V_i) > S$ ALORS élimine V_i
R3	SI V_i a un coût minimum et $S(V_i) < S$ ALORS V_i candidat pour la sélection finale

TABLE 5.8: La base de de règles de l'agent fusion

Le diagramme de cas d'utilisation ci-dessous montre les rôles de l'agent fusion.

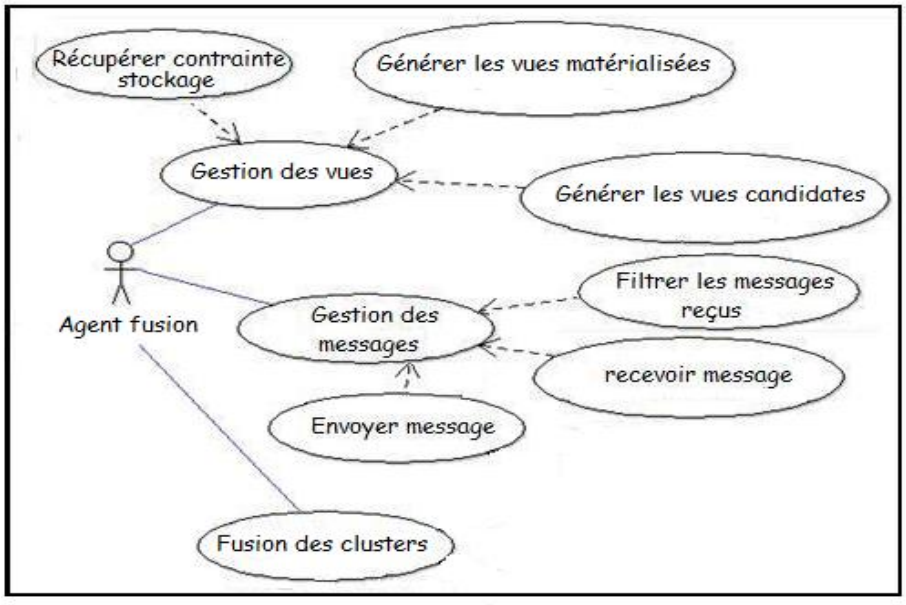

FIGURE 5.4: Diagramme de cas d'utilisation de l'agent fusion

5.4 Représentation générale de notre approche

La figure ci-dessous montre l'architecture de notre approche de sélection.

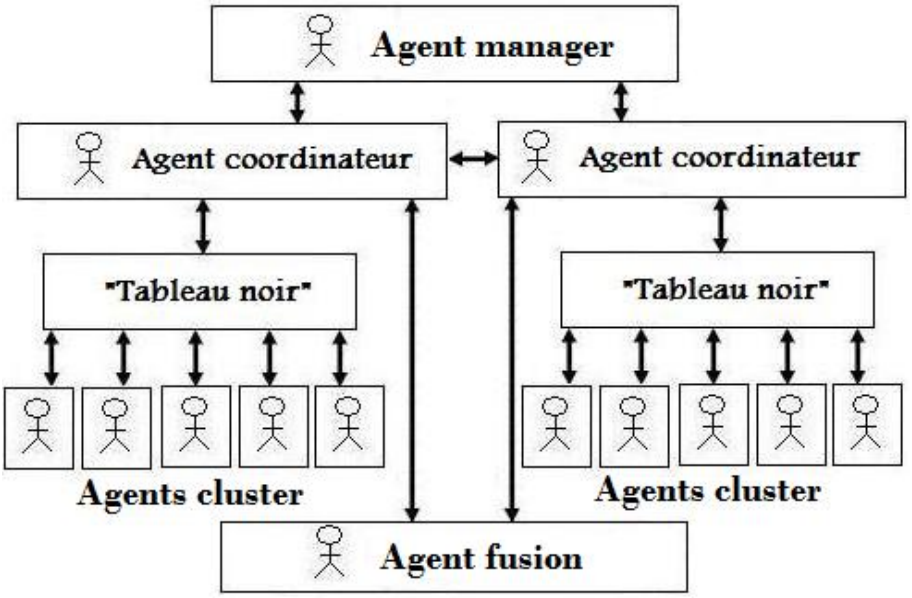

FIGURE 5.5: Architecture de notre approche de sélection

Nous remarquons qu'on a une architecture hiérarchisée modifiée composée de plusieurs niveaux. Dans ce cas, nous avons des relations entre les agents du même niveau (les agents coordinateurs). Nous avons aussi des relations via le tableau noir entre les agents de niveaux différents.

Le modèle général de notre approche est représenté par le diagramme de classes *AUML* (Figure 5.6) : une classe générique (classe Agent) est la racine de la hiérarchie des classes représentant les agents modélisés.

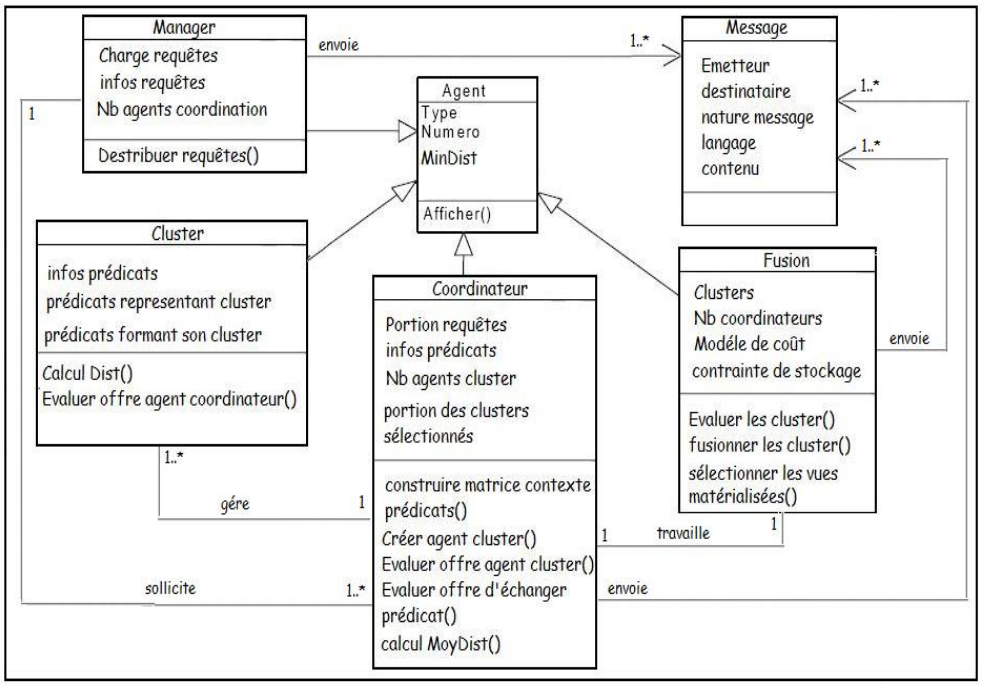

FIGURE 5.6: Diagramme de classe des agents de notre approche de sélection

5.5 Moyen et langage de communication de notre approche de sélection

Dans notre approche de sélection, nous avons adopté deux moyens de communication : le tableau noir et la communication par message. Le premier moyen a été choisi pour que les agents cluster puissent communiquer avec l'agent coordinateur de façon indirecte afin d'optimiser la communication et le nombre de contrôles effectués. Le second moyen consiste à des échanges simples de type *FIPA-ACL* entre les agents coordinateurs et l'agent manager et fusion.

5.6 Etapes de notre approche de sélection

Notre approche, considère le problème de la sélection des vues matérialisées dans le contexte d'un entrepôt de données modélisé en étoile. Elle utilise six phases principales : (1) Distribution des requêtes (2) Pré-traitement des requêtes (3) Génération des clusters (4) Optimisation de la qualité des clusters (5) Génération des vues candidates et (6) Sélection d'une configuration finale de vues à matérialisées.

5.6.1 Distribution des requêtes

Pour mettre en oeuvre cette étape, l'agent manager utilise les méta données de l'entrepôt et les informations relatives aux sources des requêtes d'interrogation pour distribuer la charge de requêtes entre les différents agents coordinateurs. Cette distribution est faite de telle façon qu'une requête ne puisse être destinée qu'à un seul agent coordinateur.

Notant que cette étape est possible car les *SGBD* actuels et notamment *Oracle* [167] et *SQLServer* [39] gardent trace des requêtes d'interrogation et des informations relatives aux utilisateurs dans des fichiers de journalisation communément appelés *log*. Par exemple, un outil tel que *Lumigent Log Explorer* [146] permet de consulter toutes les transactions déjà effectuées. Il offre un contrôle total de la base de données *SQL Server* utilisée.

5.6.2 Extraction des prédicats

Cette étape est réalisée de façon indépendante. Chaque *COA* après réception des requêtes qui lui sont destinées, sélectionne tous les prédicats qui sont référencés dans la clause *Where* de l'ensemble des requêtes que lui a transmit l'agent manager.

Une fois la phase d'extraction réalisée, le *COA* construit une matrice binaire appelée matrice de contexte des prédicats. Dans cette matrice chaque ligne et chaque colonne représentent respectivement une requête et ses prédicats. Une valeur de cette matrice d'usage des prédicats est égale à 1 si la requête utilise le prédicat, 0 sinon.

5.6.3 Génération des clusters

Cette phase réduit l'espace de recherche du problème de sélection des vues matérialisées en adaptant l'algorithme *k-means* [148], [5].

Contrairement à l'algorithme *K-means* [148], [5] où le nombre de classes doit être fixé au départ, notre démarche est incrémentale, c'est-à-dire qu'on crée au fur et mesure des agents clusters pour les prédicats qui ne respectent pas le seuil de similarité des clusters précédents.

Etant donné qu'un même prédicat ne peut exister plus d'une fois dans un même cluster, on établit que chaque agent cluster ne participera pas aux offres concernant un prédicat déjà acquit.

Cette phase arrive à son terme, si le *COA* n'a aucun prédicat à proposer et dans ce cas chaque agent cluster envoie tous les prédicats présents dans son cluster au *COA* qui l'administre. La phase de génération des clusters est subdivisée en trois étapes :

5.6.3.1 Proposition d'une offre

Cette étape débute lorsque le *COA* crée le premier agent cluster (*CA*) auquel il assigne un prédicat extrait dans la phase précédente. Notant que chaque agent cluster considère comme centre de gravité, le premier prédicat reçu.

Par la suite, en utilisant un processus itératif, le *COA* propose un prédicat aux différents *agents cluster* déjà créés et choisit l'offre la plus bénéfique.

Lors du processus d'offres, le *COA* ne propose en premier que les prédicats de sélection et ce n'est qu'une fois que ces derniers aient été distribués que le *COA* passe aux offres concernant les prédicats de regroupement et d'ordonnancement.

Le *COA* force chaque *CA* à répondre à la proposition dans un strict délai (le début et la fin de cette phase sont fixés par le *COA* qui envoie cette valeur à tous les agents cluster). Dans le cas où un agent cluster ne répond pas à l'offre dans le délai imparti, le *COA* considère ceci comme un refus. Ceci permet de limiter le nombre de communications.

5.6.3.2 Evaluation de la proposition

A la réception d'une proposition, l'agent cluster calcule la distance entre ce prédicat et son centre de gravité en utilisant la distance de Manhattan $Dist(o_i, c_j)$. Notant que la valeur $Dist(p_i, p_j)$ évaluant la distance entre un prédicat p_i et p_j est calculée comme suit :

$$Dist(p_i, p_j) = \sum_{k=1}^{m} |M(p_i, q_k) - M(p_j, q_k)| \tag{5.1}$$

Où m définit le nombre des requêtes d'interrogations contenant le prédicat $p_i$ou p_j et $M(p_i, q_k)$ représente une valeur de la matrice de contexte des prédicats.

Afin de d'éviter la construction de cluster avec des prédicats trop éloignés, un agent clusters ne considère que l'ensemble des prédicats ayant une valeur de distance inférieure ou égale à un seuil minimum, appelée $MinDist$ que l'utilisateur introduit et transmet à l'agent manager qui a son tour le transmet à l'ensemble des *COAs*.

Deux cas se présentent lors de l'envoi d'une réponse à la proposition du *COA* :

(1) un agent cluster envoie un message d'acceptation de la proposition dans le cas où la distance entre le prédicat proposé p_i et le centre de gravité c_j du cluster est inférieure au seuil $MinDist$. On a donc $Dist(p_i, c_j) \leq MinDist$.

(2) un agent cluster envoie un message de refus de la proposition dans le cas où la distance entre le prédicat proposé p_i et le centre de gravité c_j du cluster est supérieure au seuil $MinDist$. On a donc $Dist(p_i, c_j) > MinDist$.

5.6.3.3 Assignement de l'offre

Après la fin du délai imparti, le *COA* évalue les différentes offres reçues et assigne le prédicat à l'agent cluster fournissant la plus avantageuse d'entre elles (le prédicat est assigné à l'agent qui propose la plus petite des distances $Dist(p_i, c_j)$).

Dans le cas où aucune offre intéressante n'a été enregistrée, le *COA* crée un nouvel agent cluster et lui assigne le prédicat proposé. Notant enfin que dans le but de réduire la quantité d'information qui peut être échangé, nous avons supposé que quand un nouvel agent cluster rejoint le système, il ne communique qu'avec son agent coordinateur.

Notre étape de génération des clusters peut être présentée de la manière suivante :

Algorithm 5.1 Algorithme de génération des clusters

1. Le COA propose un prédicat p pour le classer
2. Chaque Agent cluster $C_i$existant fait
3. Calcule la similarité entre p et le prédicat représentatif de C_i,
4. Si $Dist(p, C_i) \leq MinDist$ Alors envoyer message d'acceptation de la proposition
5. Sinon envoyer message refus de la proposition
6. Fsi
7. Fin pour.
8. Le COA évalue les offres reçus et assigne le prédicat à l'agent cluster fournissant la plus avantageuse d'entre elles
9. Si aucune offre intéressante n'a été enregistré pour le prédicat p Alors Créer un nouveau agent cluster et lui donner o.
10. Refaire touts les étapes jusqu'à ce qu'il n'y ait plus de prédicats à proposés

5.6.4 Optimisation de la qualité des clusters

Cette phase permet de maximiser le regroupement initial par un échange de prédicats entre les différents agents coordinateurs. Ainsi, nous considérons toutes les sources d'interrogation de l'entrepôt de données. Il n'y a donc pas de contrôleur central qui dirige le comportement des agents. Ceci permet à la coopération collective d'émerger [194] parce que cet échange de prédicats permet non seulement à l'agent concerné par l'échange d'atteindre ses propres objectifs, mais ceci est aussi bénéfique à l'ensemble des autres agents.

Le diagramme de séquences ci-dessous nous permet d'avoir un aperçu sur la dynamique inter agents pour l'étape d'optimisation de la qualité des clusters.

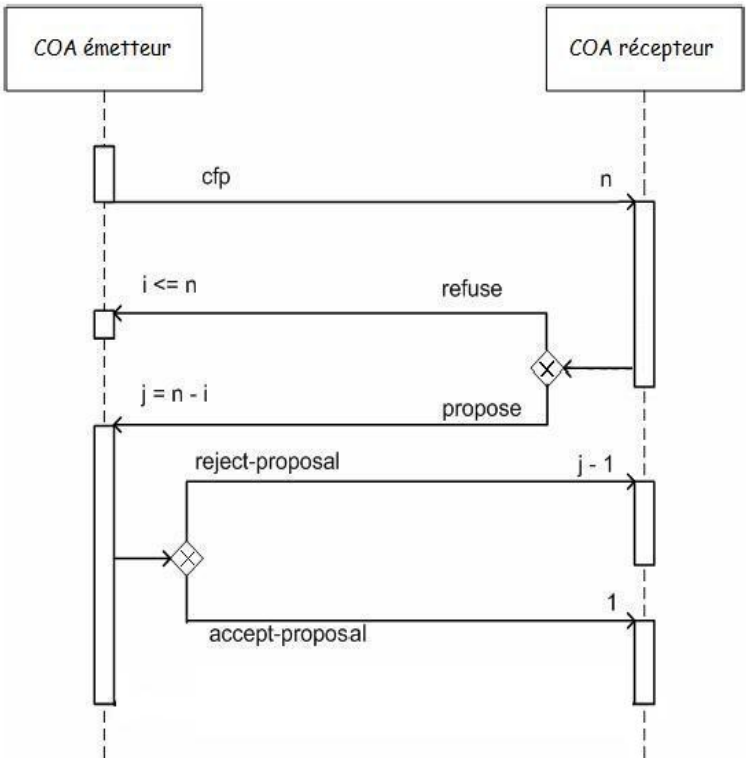

FIGURE 5.7: Diagramme de séquence de l'étape d'optimisation de la qualité des clusters

Notant que l'échange de prédicats ne concerne pas les prédicats de jointures. Ce n'est qu'une fois que cette étape soit terminée que chaque *COA* utilise la matrice de contexte pour ajouter les prédicats de jointures qui apparaissent toujours avec les éléments formant son cluster. La phase d'optimisation de la qualité des clusters est subdivisée en trois étapes :

5.6.4.1 Proposition d'une offre

Cette étape commence lorsqu'un *COA* initialise le processus d'appel d'offre avec l'envoi d'un message "annonce_échange" aux autres agents coordinateurs.

On considère une politique aléatoire dans le choix du premier agent qui va faire l'annonce. Nous considérons que ce modèle est suffisant pour capturer le comportement collectif des agents dans une population importante. En effet, [217] a montré qu'une plus grande efficacité peut être atteinte en introduisant l'attribution aléatoire dans la préférence initiale des agents.

Les échanges sont orientés dans le sens que l'annonce n'est pas envoyée à tous les autres agents [169]. Ainsi, chaque agent est autorisé à faire une soumission pour chaque annonce qu'il reçoit et les soumissions des autres ne lui sont pas révélées. En outre, chaque agent ne peut échanger les prédicats déja reçus par échange. Etant donné qu'un même prédicat ne peut exister plus d'une fois dans un même cluster, on établit que chaque *COA* ne participera pas aux offres concernant un prédicat déjà acquit.

5.6.4.2 Evaluation de l'offre

Cette étape débute après la réception du message "annonce_échange". Chaque agent coordinateur évalue cette offre en calculant la valeur moyenne $Dist(p_i, p_j)$ en considérant ces prédicats et en leurs ajoutant celui transmit par l'offre. Cette valeur est calculée comme suit :

$$MoyDist(p_i, p_j) = \frac{\sum_{k=1}^{nb} Dist(p_k, p_j)}{nb} \tag{5.2}$$

Où p_k représente chaque prédicat contenu dans le cluster et p_j celui représentant l'offre d'échange. nb représente le nombre de prédicats formant le cluster évalué.

Par la suite, il envoie sa réponse dans un strict délai (le début et la fin de ce délai sont fixés par le COA annonceur qui envoie cette valeur à tous les autres agents). Nous avons deux cas :

(1) un agent *COA* envoie un message "accept_proposal" s'il accepte l'offre. Le message "accept_proposal" est matérialisé par un prédicat qui symbolise son estimation de la demande d'échange et de ce fait son acceptation du prédicat proposé.

(2) un agent *COA* envoie un message"refuse_proposal" si l'incorporation du prédicat proposé n'améliore pas son cluster.

5.6.4.3 Réalisation de l'offre

Après la fin du délai imparti, le *COA* émetteur de l'offre évalue les différentes propositions reçues et accepte la plus avantageuse d'entre elles (le prédicat est assigné à l'agent qui propose la meilleure réduction $MoyDist(p_i, p_j)$.

Notons que le *COA* émetteur peut accepter une offre sans contre partie (pas de prédicat à échanger), dans le cas où il ne reçoit aucune autre offre plus intéressante et que l'évaluation de son cluster donne une meilleure valeur sans la présence du prédicat proposé pour l'échange. Dans le cas où aucune acceptation n'a été enregistrée, l'échange sera rejetée (aucune coopération n'est possible).

5.6.5 Construction des vues candidates.

Etant donné que notre but est la matérialisation des vues, l'agent fusion captures les dépendances en détectant les sous expressions communes au sein des différents clusters à l'aide d'une démarche adaptée de Harinarayan et al., [116].

Cette dépendance peut être détectée en premier lieu, simplement en vérifiant les *Tids* communs entre les différents clusters. En effet, deux clusters ayant une intersection vide ne peuvent être fusionnés.

Par la suite, l'agent fusion s'attache aussi à satisfaire les conditions suivantes :

- la vue obtenue par fusion doit optimiser au moins de façon similaire l'ensemble des requêtes résolues par chacune des deux vues considérées séparément,

- le coût de stockage de la vue obtenue par la fusion, doit être inférieur ou égale à celui des deux vues fusionnées.

Les clusters ainsi obtenus constitueront nos vues candidates. Notons enfin, que nous utilisons l'optimiseur des coûts du *SGBD Oracle* [166], [122] pour choisir automatiquement la plus appropriée des réécritures de chaque vue candidate.

Exemple

Nous déroulons dans ce qui suit, un exemple nous permettons d'expliciter notre démarche. Dans cet exemple nous supposons une matrice donnant les résultats obtenus en calculant les distances entre les différents prédicats préalablement extraits.

	P1	P2	P3	P4	P5	P6	P7	P8	P9
P1	0	4	3	3	4	6	6.7	6.7	9
P2	4	0	5	5	8	10	10.4	10.4	13
P3	3	5	0	6	5	6.7	6	8.4	9.4
P4	3	5	6	0	5	6.7	8.4	6	9.4
P5	4	8	5	5	0	2	3.6	3.6	5
P6	6	10	6.7	6.7	2	0	3	3	3
P7	6.7	10.4	6	8.4	3.6	3	0	6	4.2
P8	6.7	10.4	8.4	6	3.6	3	6	0	4.2
P9	9	13	9.4	9.4	5	3	4.2	4.2	0

TABLE 5.9: Matrice de proximité des prédicats

On considère dans cet exemple le seuil $MinDist = 5$.

1) Génération des clusters

Cette phase débute lorsque le *COA* crée le premier agent cluster auquel il assigne le premier prédicat $P1$. Par la suite, lors du processus itératif d'affectation, $P1$ est toujours considéré comme centre de gravité du 1er cluster (puisque c'est le premier affecté).

Par la suite le *COA* propose $P2$ qui est aussi assigné au premier agent, car la valeur $Dist(p_1, p_2) = 4$. Ce processus permet au premier agent cluster d'avoir les cinq premiers prédicats, à savoir $(P1, P2, P3, P4, P5)$. Ces prédicats ont été acceptés par rapport à leurs distances $Dist$ qui sont inférieures au seuil $MinDist$ comme nous le montre la figure suivante :

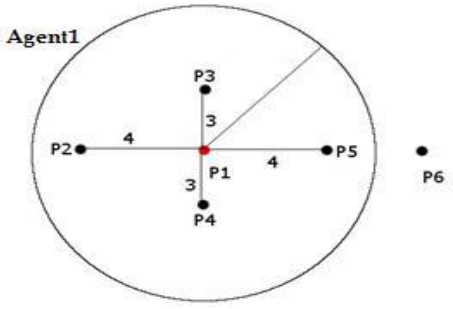

FIGURE 5.8: Notre premier cluster

A l'arrivée de $P6$, sa distance est supérieure au seuil $MinDist$ pour qu'il soit accepté par le premier agent. En effet $Dist(p_1, p_6) = 6$ et donc $Dist(p_1, p_6) > MinDist$.

L'agent coordinateur crée un deuxième agent cluster. Ce dernier considère le prédicat $P6$ comme son centre de gravité puisque c'est le premier prédicat qui lui a été affecté.

Par la suite, en utilisant le même processus itératif d'affectation, le deuxième agent cluster aura les 4 prédicats restants $(P6, P7, P8, P9)$.

La figure ci-dessous montre les résultats de notre approche après la création du deuxième cluster.

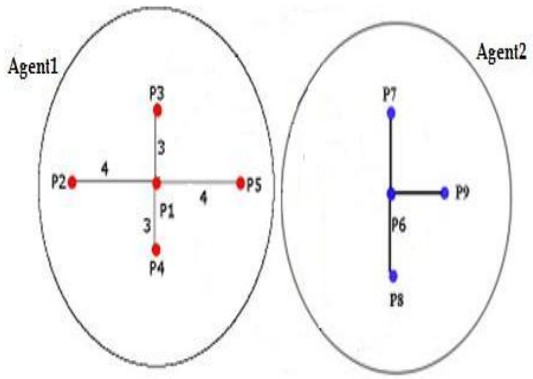

FIGURE 5.9: Résultat de l'étape de génération des clusters

5.6.5.1 *Etape de coopération*

Dans cette étape, l'agent1 propose l'échange du prédicat P5, car si on enlève l'objet $P5$, la valeur moyenne du premier cluster sera améliorée. Il passera de $\frac{14}{4}$ pour $(P1, P2, P3, P4$, avec $P5)$ à $\frac{10}{3}$ pour $(P1, P2, P3, P4)$.

Le deuxième agent accepte le prédicat $P5$, car il améliore la moyenne de son cluster qui passe de $\frac{9}{3}$ à $\frac{11}{4}$. Ainsi le prédicat $P5$ sera transféré au 2ème cluster comme dans la figure suivante :

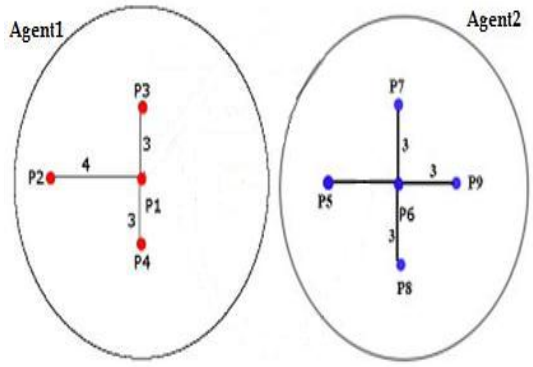

FIGURE 5.10: Résultat de l'étape de coopération

Ainsi notre démarche de sélection permet d'avoir deux vues candidates représentées par nos deux clusters.

5.7 Modèles de coût

Dans le but d'évaluer le bénéfice de la présence ou non d'une vue sélectionnée pour chaque requête, nous avons utilisé un modèle de coût qui permet de calculer les coûts nécessaires dans l'évaluation de chaque requête. Il permet aussi d'estimer la taille des vues sélectionnées.

Notre modèle de coût possède les mêmes composantes généralement utilisées par les optimiseurs [100], [152], à savoir des statistiques sur les données et des formules pour évaluer les résultats des requêtes. Nous ne considérons que le nombre d'accès au disque comme un facteur de coût à cause de son importance par rapport aux autres facteurs.

5.7.1 Bénéfice apporté à la charge par la vue

Le coût d'une vue exploitée par une requête, est égal au nombre de n-uplets que contient cette vue. Ainsi le bénéfice apporté à la charge de requêtes par l'utilisation d'une vue, est déterminé en fonction de la fréquence des requêtes qui utilisent cette vue, du poids de la vue qui est la différence entre le coût de l'ensemble des requêtes et le coût de leurs exécution en utilisant la vue. Cette valeur est donnée par la formule suivante :

$$Benefice(V) = \sum_{i=1}^{n} C(Q_i) - \lceil sup_{v_i} \times C(V) \rceil \tag{5.3}$$

où $C(Q_i)$, $C(V)$ et sup_{v_i} représentent respectivement le coût d'une requête, d'une vue et le *support* d'apparition de la vue.

5.7.2 Taille d'une vue matérialisée

Etant donné qu'une vue matérialisée est stockée sous forme de table, sa taille est donc égale au nombre de pages disque stockant cette dernière. Ainsi, la taille de l'espace de stockage requis pour une vue V, dépend du nombre de jointures qu'elle contient. Cette taille est donnée par la formule suivante :

$$S(V) = \sum_{i=1}^{n} \left\lceil \frac{|T_i|}{sel(T_i)} \right\rceil \tag{5.4}$$

où $|T|$ et $sel(T_i)$ représentent respectivement le nombre de page disque et le facteur de sélectivité de la jointure impliquant la table T.

5.8 La sélection finale de vues

La construction de toutes les vues résultantes de la dernière étape n'est pas possible. Ceci est dû notamment à la contrainte de stockage. Par conséquent, nous devons choisir parmi l'ensemble des vues proposées un sous-ensemble qui est le plus approprié pour la charge de requêtes.

Pour réaliser cette tâche, l'agent fusion utilise un modèle de coût mathématique pour évaluer chaque vue candidate (voir section précédente). Par la suite, le processus de sélection passe par deux étapes.

Dans la première, l'agent fusion rejette toutes les vues candidates et qui ne respectent pas la contrainte d'espace disque S.

Dans la seconde, l'agent fusion utilise une configuration formée par la vue qui fournit le plus d'avantage en exécutant les requêtes en tenant compte de la contrainte de stockage.

Par la suite, il améliore itérativement la configuration initiale jusqu'à ce qu'aucune réduction supplémentaire des coûts total de traitement des requêtes et aucune réduction du stockage n'est possible.

Les principales étapes de cette approche sont décrites dans Algorithme ci-dessous.

Algorithm 5.2 Algorithme de sélection finale des vues matérialisées

Entrée : $VueCand$: ensemble de vues candidates,
S : contrainte de stockage,
Sortie : $ConfigVueF$: ensemble final de vues matérialisées,

DEBUT
Evaluer le coût de chaque $V_j \in VueCand$
Tant que $(VueCand \neq \varnothing)$et $(S(V_j) > S)$ faire
$ConfigVue = ConfigVue - (V_j)$
fait ;
$ConfigF$ =Coût des requêtes sans vues
$S(ConfigVueF) = 0$
Tant que $(VueCand \neq \varnothing)$et $(S(ConfigVueF) \leq S)$ faire
Si $C(V_j) = Min(C(V_j), Q_i)$ et $C(ConfigVueF \cup V_{min}) < C(ConfigVueF)$ alors
$ConfigVueF = ConfigVueF \cup V_{min}$
$S(ConfigVueF) = S(V_{min})$
$VueCand = VueCand - (V_{min})$
finsi
Fait
FIN. // Ensemble final de vues matérialisées,

5.9 Expérimentations

Notre étude expérimentale utilise un schéma en étoile, généré par le benchmark *APB-1* [161], composé d'une table des faits *Activars* et de quatre tables de dimension : *Prodlevel*, *Custlevel*, *Chanlevel*, *Timelevel*. Le tableau suivant présente les informations concernant les tables composant notre entrepôt de données.

Paramètres	*Valeur*
$\|\|Activars\|\|$	24 786 000
$w(Activars)$	74
$\|\|Prodlevel\|\|$	10 000
$w(Prodlevel)$	72
$\|\|Custlevel\|\|$	1000
$w(Custlevel)$	24
$\|\|Timelevel\|\|$	24
$w(Timelevel)$	36
$\|\|Chanlevel\|\|$	9
$w(Chanlevel)$	24
PS	65536
π	4096

TABLE 5.10: Valeurs des tables utilisées dans notre évaluation de l'approche de sélection des vues

Le tableau ci-dessous présente les cardinalités concernant les attributs de sélection des tables de dimension utilisés dans nos évaluations expérimentales.

Table	attribut	Cardinalité
ProdLevel	Class_level	605
	Group_level	300
	Family_level	75
	Line_level	15
	Division_level	4
TimeLevel	Year_level	2
	Month_level	12
CustLevel	Retailer_level	99
ChanLevel	All_level	9

TABLE 5.11: Cardinalités des attributs de sélection utilisés dans nos évaluations de l'approche de sélection des vues

Pour valider notre approche de sélection, nous utilisons *JADE* (Java Development Framework Agent) comme outil pour mettre en œuvre notre infrastructure. Nous avons utilisé aussi une charge formée de 310 requêtes décisionnelles et six agents coordinateurs car il nous était impossible d'extraire les requêtes sans l'utilisation d'une application payante comme Lumigent's Log Explorer [146].

Nos expérimentations se sont déroulées selon trois scénarios : (1) l'évaluation de notre démarche sans considération pour l'espace de stockage, en testant les 310 requêtes décisionnelles sur l'entrepôt non optimisé, avec les vues matérialisées extraites avec notre approche

et (2) l'évaluation de notre approche avec la contrainte d'espace de stockage et (3) mesure du temps CPU lié à l'exécution de notre approche de sélection.

5.9.1 Evaluation sans contrainte d'espace

La figure 5.11 montre comment notre approche permet de réduire le coût d'exécution des 310 requêtes. Le résultat principal est que notre approche améliore les performances avec différents taux pour toutes les valeurs de *MinDist*.

La performance de notre approche se détériore pour des valeurs élevées de *MinDist*. Ceci est prévisible puisque l'augmentation de *MinDist* réduit la corrélation des prédicats formant les vues candidates.

Nous remarquons aussi que pour des valeurs très petites de *MinDist*, les performances de notre approche sont moyennes. Ceci peut s'expliquer par la présence d'un nombre élevé de vues qui doivent aussi être jointes pour réaliser l'exécution de chaque requête. En outre, nous observons aussi que le gain maximal qui dépasse les 80% est atteint pour des valeurs de *MinDist* égales à 20.

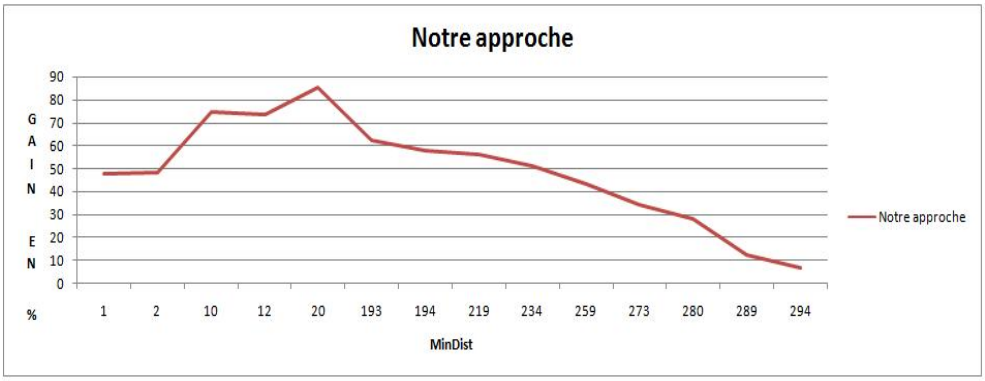

FIGURE 5.11: Qualité de notre approche de sélection des vues

5.9.2 Evaluation avec contrainte d'espace

Compte tenu de la contrainte d'espace de stockage, toutes les vues candidates ne peuvent être matérialisées dans la pratique. Pour surmonter cette limitation, nous avons utilisé l'ensemble des 310 requêtes en considérant différentes valeurs de stockage.

Nous avons aussi considéré une valeur fixe de *MinDist* égale à 20. Cette valeur permet la génération d'un grand nombre de vues candidates.

La figure 5.12 montre que, quant nous augmentons l'espace de stockage réservé aux vues, la performance s'améliore également. Ceci est prévisible parce que nous sélectionnons plus de vues lorsque l'espace de stockage augmente.

Notre approche fournit les performances les plus réduites pour des valeurs d'occupation très petites, car cette situation ne permet pas de sélectionner beaucoup de vues.

Pour conclure, notre approche nous permet de faire un meilleur choix entre les différentes vues candidates quel que soit la contrainte d'espace considérée.

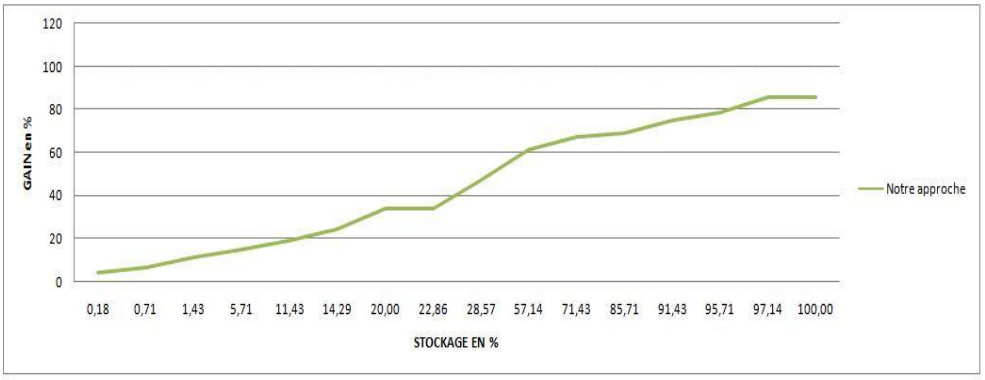

FIGURE 5.12: Evaluation avec contrainte d'espace de notre approche de sélection des vues

5.9.3 Evaluation du temps d'exécution

Nous avons effectué un autre groupe d'évaluations expérimentales concernant le temps de traitement pour notre algorithme (en microsecondes) suivant différentes valeurs de *MinDist*.

La figure 5.13 montre que l'approche nécessite des temps très variables suivant la valeur de *MinDist* que nous considérons.

Le temps de traitement nécessaire pour notre approche diminue pour les grandes valeurs de *MinDist*. En effet les grandes valeurs de *MinDist* diminuent le nombre de clusters formés.

On remarque que pour de petites valeurs de *MinDist*, notre approche nécessite un temps élevé. Ceci est dû au nombre élevé de vues extraites et par conséquent l'existence d'un nombre élevé d'agents qui communiquent entre eux dans la phase de coopération.

Pour conclure, notre approche nous permet de faire un meilleur choix entre les différentes vues candidates avec un temps d'exécution assez court si on considère une bonne valeur de *MinDist*.

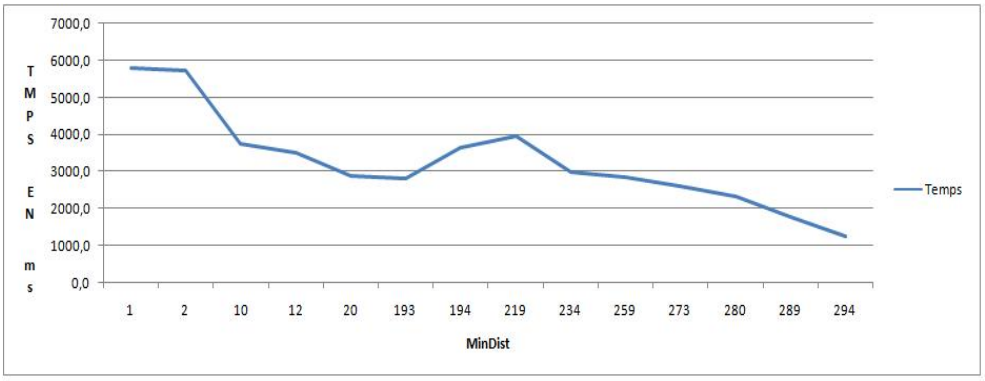

FIGURE 5.13: Evaluation du temps d'exécution de notre approche de sélection des vues

5.10 Conclusion

Dans ce chapitre, nous avons présenté notre approche de sélection des vues matérialisées dans le contexte des entrepôts de données relationnels. Afin de faire face à ce défi complexe,

nous avons utilisé deux phases principales : (1) génération des vues candidates et (2) sélection d'une configuration finale de vues matérialisées.

Dans la première phase, nous avons intégré une approche data mining avec un système multi agents afin de bien réduire l'espace de recherche dans le processus de sélection. Cette intégration a le potentiel d'offrir des avantages qui ne peuvent être obtenus en utilisant séparément l'une de ces démarches. Dans cet ordre, notre première phase apporte deux avantages :

1. En raison de sa nature distribuée, notre approche est plus efficace et plus flexible qu'une démarche centralisée,
2. En utilisant la capacité de coopération des agents, nous avons adapté un algorithme de clustering qui améliore le processus de regroupement.

Dans la deuxième phase, nous avons proposé une démarche qui sélectionne un ensemble de vues matérialisées permettant de minimiser le coût total des temps de réponse des requêtes sous une contrainte de stockage. Enfin, nous avons validé l'efficacité de notre approche en utilisant des évaluations expérimentales.

Chapitre 6

Distribution d'espace entre les index et les vues matérialisées

6.1 Introduction

Les vues matérialisées et les index sont des structures d'optimisation redondantes qui consomment de grandes quantités d'espace disque en raison des mises à jour périodiquement effectuées pour refléter les changements pouvant survenir dans les sources de données.

En pratique, le quota d'espace disque alloué aux vues et aux index est limité. Ceci oblige l'administrateur de l'entrepôt à distribuer cet espace de façon à garantir une meilleure performance pour les requêtes.

Ce chapitre nous permettra d'aborder cette problématique difficile [20], en présentant une démarche automatique de distribution d'espace disque entre les vues et les index sélectionnés de façon séparés dans les précédentes étapes.

6.2 Intuition de notre approche

Nous assimilons notre distribution à un problème de partage de ressource s'inspirant du protocole *FIPA-Contract-Net* [192], [82]. Nous considérons que notre ressource qui est l'espace de stockage peut être a priori distribuée grâce à un mécanisme d'appels d'offres émis par un agent superviseur et envoyées à deux contractants (l'agent index et à l'agent vues).

Cependant, à la différence du protocole *FIPA-Contract-Net* [192], [82], notre démarche permet non seulement à l'agent superviseur de formuler des propositions, mais donne aussi la possibilité aux contractants de formuler leurs préférences pour aboutir à une solution plus rapidement.

Notre démarche se base sur deux propriétés fondamentales qui sont :

(1) la reconnaissance de l'égalité des droits de chaque agent à prétendre à l'espace de stockage selon ces besoins,

(2) chaque agent peut prétendre à un traitement équitable de la part du superviseur.

Etant donné que dans notre cas le nombre d'agents en jeu est faible (un agent index et un agent vues) et que les agents ont des préférences simples, on préconise de considérer

le mérite comme facteur pour octroyer une portion de la ressource. Pour le superviseur, l'agent le plus méritant est celui qui fournit la structure d'optimisation (index ou vue) qui améliore le plus la configuration finale.

Notons que nous supposons ici que les parts d'espace disque sont disjointes et que la somme des quantités allouées aux agents doit être inférieure ou égale à la quantité d'espace disque disponible.

6.3 Les agents utilisés par notre approche de distribution

Notre approche considère le problème de distribution automatique, de l'espace disque disponible dans le but de fournir une performance meilleure que celle obtenue par l'ensemble des vues ou des index sélectionnés séparément. Notre approche de distribution utilise trois types d'agents.

1) Agent superviseur L'agent superviseur est doté d'un comportement cognitif. Il gère et contrôle le fonctionnement du processus de distribution et règle les conflits. Il vérifie les offres des agents et leur communique la quantité d'espace disponible.

En arbitre impartial, son objectif est l'octroie d'une portion d'espace à l'agent le plus méritant. Le tableau ci-dessous donne un aperçu de certains faits que l'agent superviseur utilise.

Fait	*Description*
F1	S l'espace de stockage alloué aux index et aux vues
F2	délai de réponse défini pour chaque offre

TABLE 6.1: La base de faits de l'agent superviseur

L'agent superviseur utilise un ensemble de règles afin d'atteindre ces objectifs. Le tableau ci-dessous donne un aperçu de certaines règles que l'agent superviseur utilise.

Règle	*Description*
R1	SI $C(BJI) \leq C(vue)$ ET $(S(BJI) \leq S_i)$ ET $(S(vue) \leq S_i)$ ALORS octroyer espace S_i à l'agent index
R2	SI $C(BJI) > C(vue)$ ET $(S(BJI) \leq S_i)$ ET $(S(vue) \leq S_i)$ ALORS octroyer espace S_i à l'agent vues
R3	SI délai offert écoulé ALORS mettre refus pour tout contractant n'ayant pas répondu
R4	SI $S(V_i) > S_i$ ALORS refuser offre agent vues
R5	SI $S(BJI) > S_i$ ALORS refuser offre agent index

TABLE 6.2: La base de faits de l'agent superviseur

Le diagramme de cas d'utilisation ci-dessous montre les rôles d'un agent superviseur.

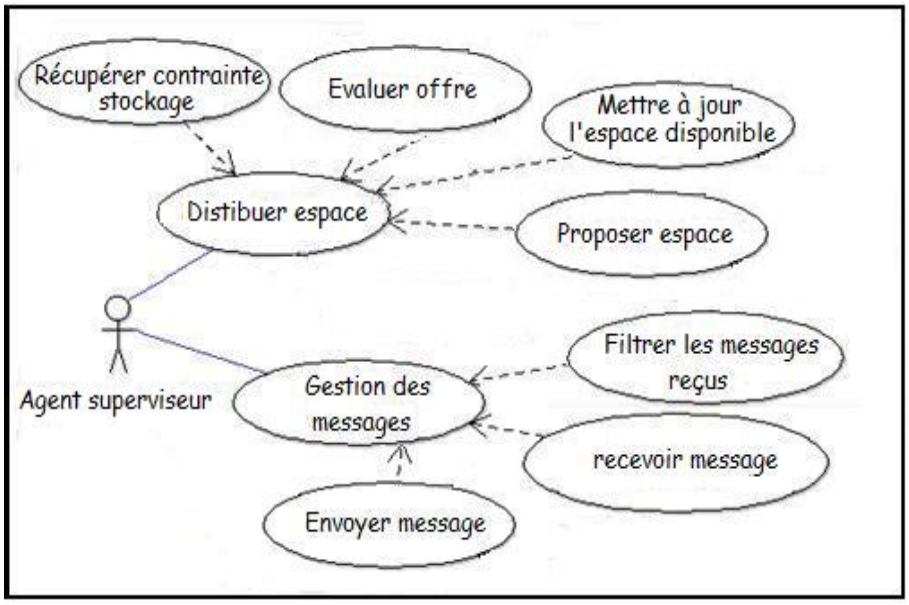

FIGURE 6.1: Diagramme de cas d'utilisation de l'agent superviseur

2. Agent index Cet agent gère l'ensemble des index obtenus à partir de l'étape de sélection des index. Il permet aussi de choisir parmi cet ensemble une configuration qu'il considère optimale pour l'espace proposé par l'agent superviseur.

L'objectif individuel d'un agent index est de faire en sorte que la configuration d'index occupe le plus d'espace disque offert. Le tableau ci-dessous donne un aperçu de certains faits que l'agent index utilise.

Fait	*Description*
F1	l'espace de stockage relatif à chaque offre
F2	délai de réponse défini pour chaque offre
F3	Modéle de coût pour évaluer les index
F4	Configuration acceptée par le superviseur

TABLE 6.3: La base de faits de l'agent index

L'agent index utilise un ensemble de règles formant sa base de connaissances. Le tableau ci-dessous donne un aperçu de certaines règles que l'agent index utilise.

Règle	*Description*
R1	SI $C(BJI(X))$ ayant la plus petite valeur) et $(S(BJI(X)) \leq$ espace offert) ALORS proposer $BJI(X)$
R2	SI $S(BJI(Y)) > S$ ALORS éliminer $BJI(Y)$
R3	SI $C(BJI(Y)) < C(vue(X))$ ET $(S(BJI(Y)) \leq S_i)$ ALORS proposer remplacement de $vue(X)$ par $BJI(Y)$

TABLE 6.4: La base de faits de l'agent index

Le diagramme de cas d'utilisation ci-dessous montre les rôles d'un agent index.

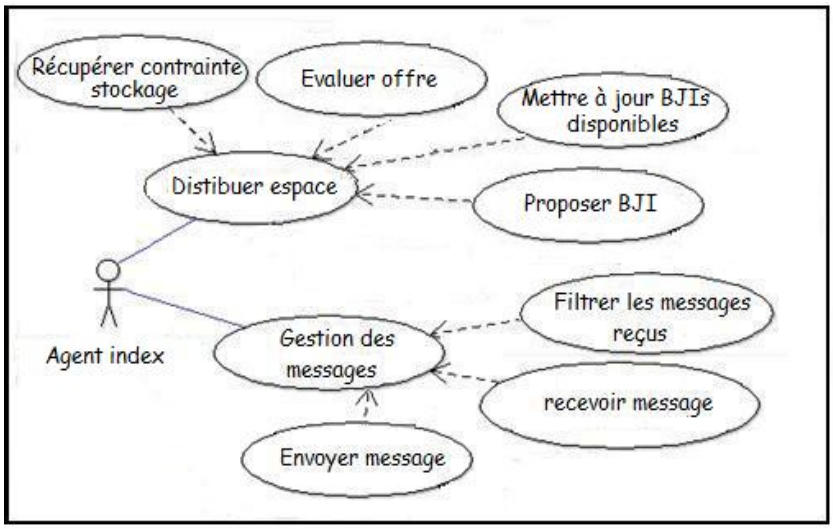

FIGURE 6.2: Diagramme de cas d'utilisation de l'agent index

3. Agent vues Cet agent a pour rôle de gérer l'ensemble des vues générées par l'étape de sélection des vues matérialisées. Il permet aussi de choisir parmi cet ensemble une configuration qu'il considère optimale pour l'espace proposé par l'agent superviseur.

L'objectif individuel d'un agent vues est de faire en sorte que la configuration des vues occupe le plus d'espace disque offert. Le tableau ci-dessous donne un aperçu de certains faits que l'agent vues utilise.

Fait	*Description*
F1	l'espace de stockage relatif à chaque offre
F2	délai de réponse défini pour chaque offre
F3	Modéle de coût pour évaluer les vues
F4	Configuration acceptée par le superviseur

TABLE 6.5: La base de faits de l'agent vues

L'agent vues utilise un ensemble de règles formant sa base de connaissances. Le tableau ci-dessous donne un aperçu de certaines règles que l'agent vues utilise.

Règle	*Description*
R1	SI $C(vue(X))$ ayant la plus petite valeur) ET ($S(vue(X)) \leq$ espace offert) ALORS proposer $vue(X)$
R2	SI $S(vue(X)) > S$ ALORS élimine $vue(X)$
R3	SI $C(BJI(Y)) < C(vue(X))$ ET ($S(vue(X)) \leq S_i$) ALORS proposer remplacement de $BJI(Y)$ par $vue(X)$

TABLE 6.6: La base de faits de l'agent vues

Le diagramme de cas d'utilisation ci-dessous montre les rôles d'un agent vues.

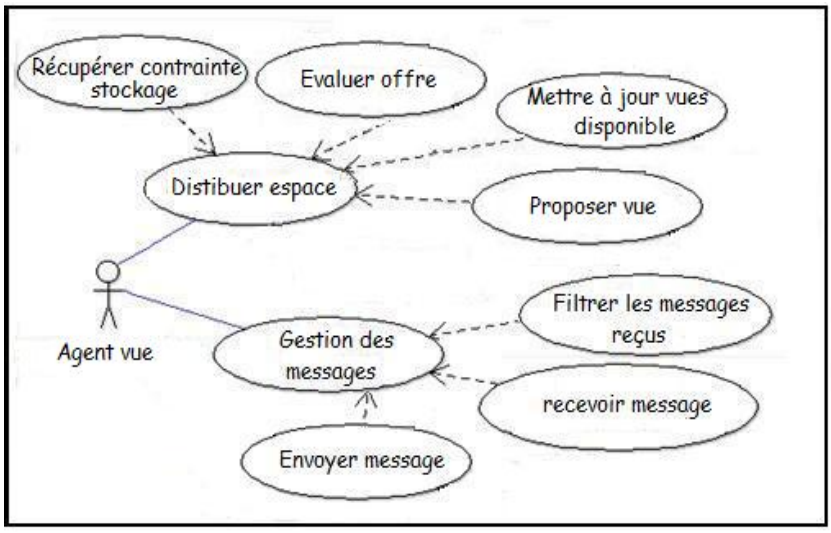

FIGURE 6.3: Diagramme de cas d'utilisation de l'agent vues

6.4 Organisation de notre approche de distribution

L'organisation est la structure décrivant comment les membres du système sont en relation afin d'atteindre un but commun en interagissant entre eux. L'organisation structurelle de notre système, est donnée par la figure suivante.

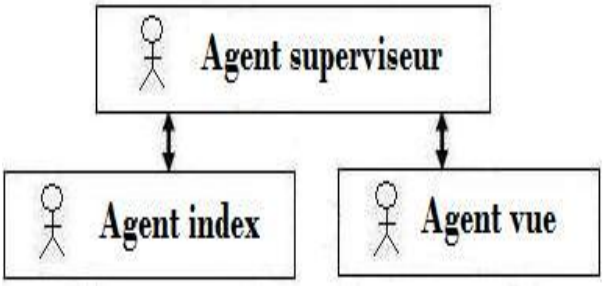

FIGURE 6.4: Architecture de notre approche de distribution

Nous remarquons qu'on a une architecture centralisée. Toutes les relations entre les agents se font via l'agent superviseur qui dispose d'une vue globale sur le système.

Le modèle général de notre approche est aussi représenté par le diagramme de classes *AUML* (Figure 6.5) : une classe générique (classe Agent) est la racine de la hiérarchie des classes représentant les agents modélisés.

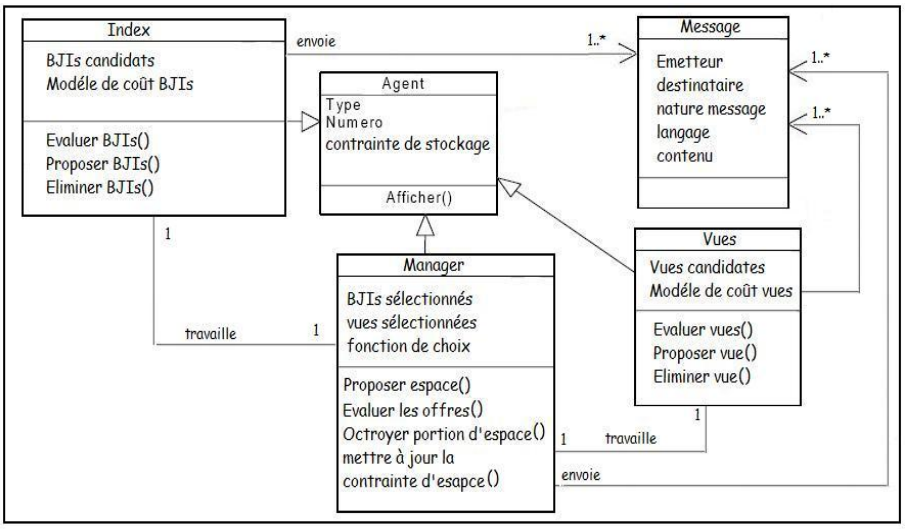

FIGURE 6.5: Diagramme de classe des agents de notre approche de distribution

6.5 Moyen et langage de communication de notre approche de distribution

Etant donné que le nombre d'agents utilisés dans notre approche de distribution, est réduit. Nous avons adopté un moyen de communication basé sur des échanges simples de type *FIPA-ACL* entre l'agent superviseur et l'agent index et vues.

Voici un exemple de message cfp (Call For Proposal) issue de [81]. Dans ce message $l'agent_j$ propose à $l'agent_i$ la vente d'une voiture avec un prix de 5.000 euro.

(cfp
:sender (agent-identifier : name i)
:receiver (set(agent-identifier :name j))
:content prix(clio02,5000 EUR))

:ontology voiture
:langage fipa-sl)
:reply-by 1 min)

6.6 Etapes de notre approche de distribution

Notre approche utilise deux étapes principales. La première permet d'obtenir une distribution initiale. Cette dernière sera améliorée de façon itérative dans la seconde étape.

Notons que les deux agents contractants (agent index et agent vues) utilisent respectivement les deux configurations candidates (index et vues) qu'ils évaluent respectivement grâce aux deux modèles de coût précédemment cités (voir chapitre 4 et 5).

Notant aussi que lors de chaque étape de notre processus de distribution, les messages envoyés par les contractants sont uniquement destinés au superviseur. Les contractants ne connaissent pas la liste des participants conviés au processus de distribution, ils ne peuvent donc pas former de coalition au cours de la négociation.

6.6.1 Construction de la configuration initiale

Dans cette étape, l'agent superviseur procède à un ensemble d'appel d'offres. Chaque appel d'offre concerne les deux contractants (l'agent index et l'agent vues) pour une portion d'espace disque. Ce processus s'arrête lorsqu'aucune offre n'est possible (aucun index ou vue ne permet d'améliorer un peu plus la configuration déjà sélectionnée ou aucune réduction du stockage n'est possible).

Le diagramme de séquences ci-dessous nous permet d'avoir un aperçu sur la dynamique inter agents pour l'étape de construction de la configuration initiale.

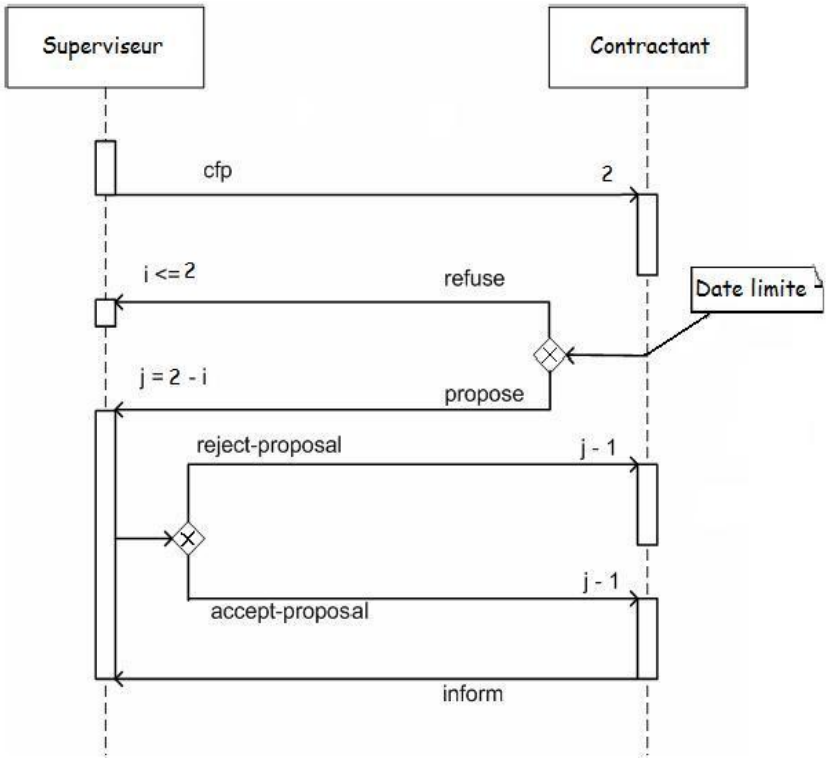

FIGURE 6.6: Diagramme de séquence de l'étape de construction de la configuration initiale

6.6.1.1 Lancement de l'appel d'offre

Cette étape commence lorsque l'agent superviseur envoie aux deux agents contractants une demande d'octroie d'une structure d'optimisation (index ou vue) respectant la contrainte d'espace disque.

Dans cette étape, seul le superviseur émet des offres, les contractants ne peuvent que faire une proposition et pas de contre-proposition.

L'agent superviseur force chaque agent contractant à répondre à l'offre dans un strict délai (le début et la fin de cette phase sont fixés par le superviseur qui envoie cette valeur aux deux agents contractants).

Dans le cas où un contractant ne répond pas à l'appel d'offre dans le délai imparti, le superviseur considère ceci comme un refus. Ceci permet de limiter le nombre de communications.

6.6.1.2 Proposition d'offre

A cette étape, chaque agent procède de façon indépendante. Il n'y a pas de synchronisation lors du choix de chaque agent.

Pour fournir une configuration valide, l'agent index et vues utilisent deux étapes. Dans la première, ils rejettent respectivement toute structure (index ou vue) qui ne respecte pas la contrainte d'espace disque alloué par le superviseur.

Par la suite, l'agent index utilise l'évaluation de la configuration des candidats pour choisir l'index qui améliore le plus la charge de requêtes.

De façon similaire, l'agent vues utilise l'évaluation de la configuration des vues candidates pour choisir celle qui fournit le plus d'avantage à la charge de requêtes. L'agent contractant possédant une offre envoie celle-ci à l'agent superviseur.

6.6.1.3 Attribution de l'offre

Cette étape commence lorsque l'agent superviseur reçoit un index et une vue respectant la contrainte d'espace disque. Dans le but de former une unique configuration l'agent superviseur compare les performances de l'index et de la vue et prend la structure (index ou vue) dont l'intégration améliore encore plus la configuration déjà acquise.

Par la suite, l'agent superviseur transmet un message d'acceptation à celui qu'il choisit et un message de rejet à l'autre agent. Enfin, l'agent superviseur met à jour l'espace de stockage.

L'étape de construction de la configuration initiale peut être décrite par l'algorithme suivant :

Algorithm 6.1 Algorithme décrivant la premiere étape de distribution d'espace entre les index et les vues

Entrées : $ConfigBJI = \{I_1, I_2, ...I_m\}$ ensemble d'index
$ConfigVM = \{V_1, V_2, ...V_K\}$ ensemble de vues
S (espace de stockage)

Debut
$S(Config) = 0$
Tant que $(S(Config) < S)$ faire
1. L'agent superviseur envoie, aux deux agents contractants, un message «annonce-proposition» contenant la taille d'espace S,
2. Chaque agent contractant évalue la proposition,
3. Si $S(I_{min}) \leqslant S$ alors agent index soumet I_{min} à l'aide du message «proposition-index» au superviseur,
3. Si $S(V_{min}) \leqslant S$ alors agent vues soumet V_{min} à l'aide du message « proposition-vue » au superviseur,
4. L'agent superviseur évalue les propositions reçues
4.1 Si $C(ConfigF \cup V_{min}) < C(ConfigF)$ et $C(V_{min}) < C(I_{min})$ alors
4.1.1) envoie un message «acceptation» à l'agent vues
4.1.2) envoie un message «refus» à l'agents index
4.2 Si $C(ConfigF \cup I_{min}) < C(ConfigF)$ et i $C(I_{min}) < C(V_{min})$ alors
4.2.1) envoie un message «acceptation» à l'agent index
4.2.2) envoie un message «refus» à l'agents vue
4.3 Il met à jour l'espace disque libre,
4.3.1 Si I_{min} est retenu alors $S(ConfigF) = S(ConfigF) + S(I_{min})$
4.3.2 Si V_{min} est retenue alors $S(ConfigF) = S(ConfigF) + S(V_{min})$
5. Si le temps d'expiration est dépassé alors l'agent superviseur met fin à la période d'acceptation de proposition
6. Si aucune proposition n'a été retenue, alors le superviseur envoie aux deux contractants un message "refus" pour indiquer le rejet de chacune des propositions,
7. L'agent ayant été retenu met à jour sa configuration.
7.1.1 Si I_{min} est retenu alors $ConfigBJI = ConfigBJI - (I_{min})$
7.1.2 Si V_{min} est retenue alors $ConfigVM = ConfigVM - (V_{min})$
Fait ;
Fin.

6.6.2 Affinement de la configuration initiale

Cette étape permet d'affiner de façon itérative la solution précédemment obtenue. Dans ce but, l'agent superviseur initialise ce processus en envoyant, aux deux agents contractants, une demande d'amélioration de configuration obtenue. Chaque agent procède de façon indépendante. Il n'y a pas de synchronisation lors du choix de chaque agent.

L'agent superviseur force chaque agent contractant à répondre à la proposition dans

un strict délai (le début et la fin de cette phase sont fixés par le superviseur qui envoi cette valeur aux deux agents contractants).

Dans le cas où un contractant ne répond pas à l'appel d'offre dans le délai imparti, le superviseur considère ceci comme un refus. Ceci permet de limiter le nombre de communications.

Afin de faire la meilleure proposition possible, l'agent vues essaye de trouver un index dont le remplacement par une des vues qu'il possède permet une plus grande amélioration du coût d'exécution des requêtes. La configuration ainsi obtenue doit aussi respecter la contrainte d'espace transmise par le superviseur.

D'une manière similaire, l'agent index essaye de trouver une vue dont le remplacement par un des index qu'il possède permet une plus grande amélioration du coût d'exécution des requêtes. La configuration ainsi obtenue doit aussi respecter la contrainte d'espace transmise par le superviseur.

Dans le but de former la nouvelle distribution, l'agent superviseur compare les performances des propositions reçues et accepte celle (index ou vue) dont l'intégration améliore encore plus la configuration déjà acquise. Par la suite, il met à jour l'espace de stockage.

L'algorithme s'achève lorsque les deux agents ne peuvent plus réduire le coût total d'exécution des requêtes. Cette étape peut être décrite par l'algorithme suivant :

Algorithm 6.2 Algorithme décrivant la deuxieme étape de distribution d'espace entre index et vues

Entrées : $ConfigF = \{I_1, I_2, ...I_l, V_1, V_2, ...V_j\}$ ensemble d'index et de vues obtenues par la premiere étape

$ConfigBJI = \{I_1, I_2, ...I_m\}$ ensemble d'index candidats

$ConfigVM = \{V_1, V_2, ...V_K\}$ ensemble de vues candidates

S (espace de stockage)

Sortie : $ConfigF = \{I_1, I_2, ...I_h, V_1, V_2, ...V_K\}$ nouvelle ensemble d'index et de vues

Debut

Tant que contractants $(S(ConfigF) < S)$ faire

1. L'agent superviseur envoie, aux deux agents contractants, un message «annonce-amélioration» contenant la taille d'espace S,

2. Chaque agent contractant évalue la proposition,

2.1. *Si* $(\ni \ I_i \in ConfigF)$ et $[C((ConfigF - V_i) \cup I_j) < C(ConfigF)]$ et $[S((ConfigF - V_i) \cup I_j) \leq S]$ *alors* agent index soumet I_j à l'aide du message «proposition-index» au superviseur,

2.2. *Si* $(\ni \ V_i \in ConfigF)$ et $[C((ConfigF - I_i) \cup V_j) < C(ConfigF)]$ et $[S((ConfigF - I_i) \cup V_j) < S]$ *alors* agent vues soumet au superviseur V_j à l'aide du message «proposition-vue»,

3. L'agent superviseur évalue les propositions reçues

3.1 Si $[C((ConfigF - I_i) \cup V_j) < C(ConfigF)]$ et $[S((ConfigF - I_i) \cup V_j) \leq S]$ *alors*

3.1.1) Il envoie un message «acceptation» à l'agent vues

3.1.2) Il envoie un message «refus» à l'agents index

3.2 Si $[C((ConfigF - V_i) \cup I_j) < C(ConfigF)]$ et $[S((ConfigF - V_i) \cup I_j) \leq S]$ *alors*

3.1.1) Il envoie un message «acceptation» à l'agent index

3.1.2) Il envoie un message «refus» à l'agents vue

4) Il met à jour l'espace disque libre,

4.1) Si I_j est retenu alors$S(ConfigF) = [S(ConfigF) - S(V_i)] + S(I_j)$

4.2) Si V_i est retenue alors $S(ConfigF) = [S(ConfigF) - S(I_j)] + S(V_i)$

5. Si le temps d'expiration est dépassé alors l'agent superviseur met fin à la période d'acceptation des propositions

6. Si aucune proposition n'a été retenue, alors le superviseur envoie aux deux contractants un message "refus" pour indiquer le rejet de chacune des propositions,

7. L'agent ayant été retenu met à jour sa configuration.

4.3.1 Si I_i est retenu alors$ConfigBJI = ConfigBJI - (I_i)$

4.3.2 Si V_j est retenue alors$ConfigVM = ConfigVM - (V_j)$

Fait ;

Fin.

6.7 Expérimentations

Notre étude expérimentale utilise un schéma en étoile, généré par le benchmark APB1 [161], composé d'une table des faits *Activars* et de quatre tables de dimension : *Prodlevel*, *Custlevel*, *Chanlevel*, *Timelevel*. Nous gardons les mêmes valeurs concernant les tables et les attributs que ceux précédemment cités aux chapitre quatre et cinq.

Notons enfin, que pour valider notre approche de distribution, nous utilisons *JADE* (Java Development Framework Agent) comme outil pour mettre en oeuvre notre infrastructure.

6.7.1 Evaluation de notre approche de distribution

La figure 6.7 nous montre l'évaluation du gain apportée par notre approche avec différentes contraintes d'espace de stockage. Nous considérons l'évaluation des 310 requêtes de la charge sans optimisation, avec index, avec vues et avec index et vues matérialisées en utilisant notre approche de distribution.

Nous avons aussi considéré une valeur fixe de *MinDist* et *MinWeight* respectivement égale à 20 et 0,0000106319. Ces deux valeurs permettent la génération d'un grand nombre de vue et d'index candidats.

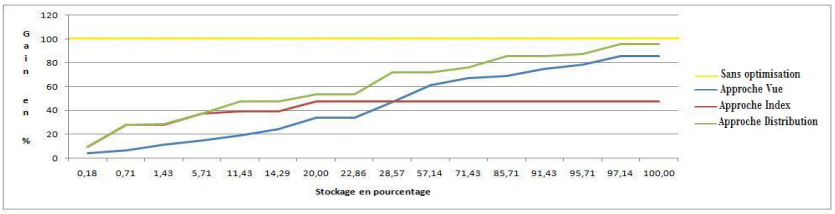

FIGURE 6.7: Evaluation de notre approche de distribution

La Figure 6.7 montre que notre approche de distribution offre les meilleures performances pour les différentes valeurs de stockage considérées.

Nous remarquons aussi que pour les valeurs élevées de l'espace de stockage, l'approche de distribution offre de meilleures performances que la sélection isolée des index et vues.

Par contre, nous constatons que pour les petites valeurs de l'espace de stockage, il peut arriver que l'approche de distribution offre les mêmes performantes que la sélection d'index. Cela peut être expliqué par le fait qu'en général, la taille des index est significativement moins importante que celle des vues. Dans ce cas, notre approche de distribution sélectionne plus d'index que de vues.

6.8 Conclusions

La sélection des index et des vues permet de réduire le coût d'évaluation des requêtes. Etant donné que ces deux structures redondantes partagent le même espace de stockage, une distribution automatisée de cet espace s'avère primordiale.

Dans ce chapitre, nous avons développé une nouvelle méthode de distribution de l'espace entre les vues et les index dans le contexte statique. Nous avons assimilé notre distribution à un problème de partage de ressource s'inspirant du protocole *FIPA-Contract-Net*.

Notre démarche permet aux agents concernés d'aboutir de manière décentralisée à un partage de l'espace de stockage qu'on considère le plus optimal possible. Afin d'atteindre notre objectif, nous avons utilisé un agent superviseur comme arbitre impartial et considéré le mérite comme facteur d'octroie d'une portion de la ressource.

A la différence du protocole *FIPA-Contract-Net*, nous avons permis non seulement à l'agent superviseur de formuler des propositions, mais aussi donné la possibilité aux agents contractants de formuler leurs préférences pour aboutir à une solution plus rapidement. Les évaluations expérimentales qu'on a effectué ont montré la justesse de nos choix et explicité les qualités de notre approche.

Conclusion et perspectives

Cette thèse a concerné les techniques redondantes pour l'optimisation de l'administration des entrepôts de données. Nous nous sommes intéressés aussi à la distribution d'espace entre les index et les vues matérialisées.

1. Contributions

Dans cette section nous résumons les principaux apports de nos travaux.

1.1 Sélection d'une configuration d'index de jointure binaires

La sélection d'index est un problème très difficile. Nous avons formulé ce dernier comme un problème d'optimisation minimisant une fonction de coût d'exécution des requêtes en présence d'une contrainte d'espace.

Nous avons choisi de nous focaliser sur la sélection des index de jointure binaires puisqu'ils sont bien adaptés à l'environnement des entrepôts de données. Cependant, notre stratégie de sélection d'index reste tout à fait facile à adapter à d'autres techniques d'indexation par quelques modifications qui toucheront la phase d'épuration des itemsets extraits et d'appliquer les modèles de coût correspondant à ces index.

Notre démarche de sélection se base sur deux phases : (1) la sélection des index candidats et (2) la sélection d'une configuration finale d'index. La phase de sélection des index candidats nous permet de réduire l'espace de recherche en éliminant les attributs non pertinents. Pour correctement élaguer l'espace de recherche des index, nous avons en premier lieu montré les principales limitations des précédents travaux.

Afin de contribuer à pallier ces insuffisances, nous avons proposé une démarche data mining distribuée à base d'un système multi agents coopératif. Cette intégration nous a offert une plus grande flexible qu'une démarche centralisée,

En utilisant la capacité de coopération des agents, nous avons adapté un algorithme de recherche de motifs fréquents fermés pour réduire l'espace de recherche des index candidats.

Contrairement aux approches classiques qui utilisent uniquement la fréquence d'apparition des itemsets comme critère d'élagage, nous avons enrichis ce critère en prenant en compte un facteur pénalisant relatif à la cardinalité et au nombre de n-uplet des différentes tables de dimension par rapport à la table des faits. Ce facteur change dynamiquement lors de l'exécution de l'algorithme d'extraction des itemsets.

Pour la phase de sélection d'une configuration finale d'index, nous avons aussi proposé une autre stratégie utilisant un système multi agents permettant de sélectionner sous une

contrainte de stockage, les index les plus avantageux grâce à un modèle de coût mathématique.

Les résultats de nos expérimentations ont montré l'efficacité de notre approche de sélection et sa contribution significative à la réduction du temps d'exécution d'un ensemble de requêtes.

1.2 Sélection d'une configuration de vues matérialisées

Afin de faire face au problème de sélection des vues matérialisées dans le contexte des entrepôts de données relationnels, nous avons proposé une approche distribuée de classification non supervisée et utilisant un système multi agents coopératif.

Notre démarche de classification distribuée permet de réduire la complexité de résolution de ce problème tout en regroupant de façon efficace les différents prédicats formant nos vues candidates. En effet, les mesures de distance qu'on a utilisée dans le processus de classification permettent de capturer les relations qui peuvent exister entre les différents prédicats pour les regrouper dans les vues candidates.

Notre démarche de construction de la configuration finale des vues matérialisées permet de minimiser le coût total des temps de réponse des requêtes sous une contrainte de stockage.

A cet effet, nous avons préconisé un agent qui estime les vues candidates grâce à un modèle de coût qui évalue le nombre d'entrées sorties nécessaires à l'exécution d'un ensemble de requêtes. Ceci permet en premier lieu d'éliminer les vues ne respectant pas la contrainte d'espace. Par la suite l'agent choisi parmi les vues les plus performantes par rapport à l'amélioration du temps de réponses des requêtes. Un ensemble d'évaluations expérimentales nous a permis de valider l'efficacité de notre approche.

1.3 Distribution d'espace entre les vues et les index

Les index et les vues matérialisées sont deux structures redondantes qui sont souvent utilisées ensemble et partagent le même espace disque. Afin de permettre un partage efficace de l'espace de stockage disponible entre ces deux structures, nous avons suggéré une nouvelle approche pour distribuer l'espace entre les vues et les index dans le cas statique. Nous avons assimilé notre distribution à un problème de partage d'une ressource s'inspirant du protocole *FIPA-Contract-Net*.

Notre démarche permet aux agents concernés d'aboutir de manière décentralisée à un partage de l'espace de stockage qu'on considère le plus optimal possible. Afin d'atteindre notre objectif, nous avons utilisé un agent superviseur comme arbitre impartial et considéré le mérite comme facteur d'octroie d'une portion de la ressource.

A la différence du protocole *FIPA-Contract-Net*, nous avons permis non seulement à l'agent superviseur de formuler des propositions, mais aussi donné la possibilité aux agents contractants de formuler leurs préférences pour aboutir à une solution plus rapidement. Nos évaluations expérimentales ont montré que notre démarche de distribution permet d'améliorer encore plus les performances des requêtes d'interrogation.

2. Perspectives

Le travail réalisé dans cette thèse ouvre la voie à de nombreuses extensions et perspectives. Dans cette section nous présentons celles qui nous paraissent être les plus importantes.

2.1 Sélection dynamique des index et des vues

Dans ce travail on s'est focalisé sur la sélection des structures d'optimisation redondantes dans le cas statique. Il serait intéressant de considérer des études complémentaires pour prendre en compte les effets des opérations de mise à jour des données de l'entrepôt sur la qualité des index et des vues. Ceci peut être réalisé par une approche permettant de changer automatiquement un certain nombre de vues ou d'index.

Dans ce cadre, on pourrait associer des statistiques sur les requêtes indiquant la fréquence avec laquelle les vues matérialisées ou les index sont utilisés. Cette information peut être utilisée pour supprimer les vues matérialisées ou les index qui sont rarement utilisés. Elle pourra servir aussi pour procéder à une nouvelle sélection pour améliorer l'exécution de requêtes lentes.

2.2 Considération d'autres contraintes d'évaluation

Notre approche considère l'espace disque comme seule contrainte à respecter. D'autres contraintes peuvent être envisagées comme par exemple le coût de maintenance des index et des vues. En effet, la prise en compte d'autres facteurs comme le coût de maintenance, la présence d'index sur les vues et l'évolution des requêtes tant dans leurs structures que dans leurs fréquences, peuvent grandement influencer l'évaluation des performances.

2.3 Sélection des structures d'optimisation non redondantes

Une autre direction d'enrichissement de l'approche concerne son extension pour traiter simultanément le problème de sélection des structures d'optimisation redondantes et non redondante.

En effet, on pourrait mettre en place une démarche de fragmentation et par la suite procéder à la sélection des index et des vues matérialisées sur le schéma fragmenté. Cette démarche aura l'avantage d'améliorer un peu plus le temps d'exécution des requêtes.

2.4 Détermination de la fréquence de maintenance

Une autre étude intéressante serait de déterminer la fréquence optimale à laquelle une opération de maintenance d'une vue doit être effectuée.

Serait-il plus efficace d'exécuter la maintenance fréquemment au cours de petites modifications des données ? Ou est-ce qu'il vaut mieux accumuler les changements sur une période significative de temps avant l'exécution de la maintenance ?

2.5 Amélioration de l'efficacité des agents

Nous pourrons encore augmenter l'efficacité des agents en leur permettant de sélectionner, en cas de présence de plusieurs possibilités, la stratégie la plus pertinente. Donc ce cas de figure on pourrait utiliser la plus réalisable et la plus efficace suivant la situation.

Ces stratégies pourraient consister à considérer d'autres types d'algorithmes et d'autres heuristiques pour la réduction de l'espace de recherche.

2.6 Distribution d'espace à base d'un modèle de négociation améliorée

Une autre stratégie de distribution d'espace entre l'agent des index et des vues, peut se baser sur les contraintes de négociation et utilisant un modèle pour sélectionner les arguments les plus pertinents.

À cet effet, en utilisant les contraintes de négociation, chaque agent est capable de déterminer son espace d'accord. Ainsi, chaque agent pourrait calculer son degré de satisfaction, l'ensemble de ses arguments et de ses offres potentielles. Ainsi chaque agent essayera de réaliser son but en utilisant des arguments pour convaincre l'autre agent de faire des concessions.

Bibliographie

[1] R Agrawal, C Aggarwal, and V. V Prasad. Depth first generation of long patterns. *In 7th International Conference on Knowledge Discovery and Data Mining*, pages 108–118, 2000.

[2] R Agrawal, C Aggarwal, and V. V Prasad. A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, 61(3) :350–371, 2001.

[3] R Agrawal, A Gupta, and S Sarawagi. Modeling multidimensional databases. Technical report, Research Report, IBM Almaden Research Center, San Jose, USA, 1996.

[4] R Agrawal and R Srikant. Fast algorithms for mining association rules in large databases. *20th international conference on Very Large Data Bases (VLDB 1994)*, pages 478–499, 1994.

[5] M. R Anderberg. Cluster analysis for applications. *Academic press*, 1973.

[6] P Andritsos. Data clustering techniques. *qualifying oral examination paper*, page 2002.

[7] K Aouiche. *Techniques de fouille de donnée pour l'optimisation automatique des performances des entrepôts de données.* PhD thesis, Université Lumière Lyon2, France, 2005.

[8] K Aouiche and J Darmont. Data mining based materialized view and index selection in data warehouses. *Journal of intelligent information systems, springer*, Number 1, Vol 33, 2009.

[9] K Aouiche, J Darmont, O Boussaid, and F Bentayed. Automatic selection of bitmap join indexes in data warehouses. *7th International Conference on Data Warehousing and Knowledge Discovery*, pages 64–73, 2005.

[10] K Aouiche, P.E Jouve, and J Darmont. Clustering based materialized view selection in data warehouses. *Lecture notes in computer science*, 4152 :81–95, 2006.

[11] L Audibert. *UML 2, de l'apprentissage à la pratique.* Ellipses, 2009.

[12] S Azefack, K Aouiche, and J Darmont. Dynamic index selection in data warehouses. *4th International Conference on Innovations in Information Technology*, 2006.

[13] E Baralis, S Paraboschi, and E Teniente. Materialized views selection in a multi dimensional database. *In 23rd international conference on very large data bases*, pages 156–165, 1997.

[14] Y Bastide, N Pasquier, R Taouil, L Lakhal, and G Stumme. Mining minimal non redundant association rules using frequent closed itemsets. *Proceedings of the international conference DOOD 2000, LNCS, Springer-verlag*, pages 972–986, 2000.

[15] Y Bastide, R Taouil, N Pasquier, Stumme G, and L Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explorations, ACM Computer*, 2(2) :66–75, 2000.

[16] Y Bastide, R Taouil, N Pasquier, G Stumme, and L Lot. Pascal : un algorithme d'extraction des motifs fréquents. *Techniques et Science Informatiques*, 21(1) :65–95, 2002.

[17] B Bauer. Uml class diagrams revisited in the context of agent based systems. *Proceedings of the second international workshop on agent oriented software engineering, springer*, pages 101–118, 2001.

[18] B Bauer, J.P Muller, and J Odell. Agent uml : a formalism for specifying multi agent interaction. *Agent oriented software engineering, springer-verlag*, pages 91–103, 2001.

[19] R.J Bayardo. Efficiently mining long patterns from databases. *Proceedings of the international conference on management of data (ACM SIGMOD)*, 1998.

[20] L Bellatreche. Utilisation des vues matérialisées, des index et de la fragmentation dans la conception logique et physique d'un entrepôt de données. *Thèse de Doctorat. Université de Clermont-ferrant II. France*, 2000.

[21] L Bellatreche, R Missaoui, H Necir, and H Drias. Selection and pruning algorithms for bitmap index selection problem using data mining. *In the 9th international conference on data warehousing and knowledge discovery, edited by lncs*, pages 221–230, 2007.

[22] L Bellatreche, R Missaoui, H Necir, and H Drias. A data mining approach for selecting bitmap join indices. *Journal of computing science and engineering*, Volume 2, Num 1 :206–223, 2008.

[23] S Benyahia, C. L Cherif, G Mineau, and Jaoua A. Découverte des règles associatives non redondantes : application aux corpus textuels. *Revue d'intelligence artificielle (EGC 2003)*, 17, no 1-2-3 :131–143, 2003.

[24] S Benyahia and E Mephu Nguifo. Approches d'extraction de règles d'association basées sur la correspondance de galois. *Ingénierie des systèmes d'information*, vol. 9, no 3-4 :23–55, 2004.

[25] P Berkhin. Survey of clustering data mining techniques. technical report. Technical report, Accrue Software, 2002.

[26] M Berna-koes, I Nourbakhsh, and K Sycara. Communication efficiency in multi agent systems. *In proceedings of IEEE international conference on robotics and automation*, 2004.

[27] J Bezdek, R Haihaway, M Sabin, and W Tucker. Convergence theory for fuzzy c-means : counter examples and repairs. *Institute of electrical and electronic engineers transactions on systerns, man and cybernetics*, 17 :873–877, 1987.

[28] O Boissier, S Gitton, and P Glize. Caractéristiques des systèmes et des applications. *In systèmes multi agents, observatoire français des techniques avancées*, pages 25–54, 2004.

[29] F Bonchi. *Frequent pattern queries : languages and optimizations.* PhD thesis, Università di Pisa TD, 2003.

[30] A. H Bond and L Gasser. Readings in distributed artificial intelligence. *Morgan Kaufmann Publishers. San Mateo. CA.*, 1988.

[31] HS Botre, MS Chaudhari, and SB Chaturvedi. Materialized view selection algorithm : a survey. *International journal of engineering research and technology*, 1, issue 10, 2012.

[32] J-F Boulicaut and B Jeudy. Using constraints during itemset mining : should we prune or not ? *International conference of BDA'00*, pages 221–237., 2000.

[33] T Bouront. *Structures de communication et d'organisation pour la coopération dans un univers multi-agents.* PhD thesis, Université pierre et marie curie (Paris 6). France, 1992.

[34] S.P Bradley, U Fayyad, and C Reina. Scaling em (expectation maximization) clustering to large databases. Technical report, Microsoft research, 1999.

[35] C Brassac and S Pesty. Coopération dans les systèmes multi agents : Comportement ou conduite ? *Proceedings of the 1st international workshop on decentralized intelligent and multi agent systems*, pages 84–92, 1995.

[36] M Bratman. *Intention, plans and practical reason.* Csli publications, 1987.

[37] S Brin, R Motwani, J. D Ullman, and S Tsur. Dynamic itemset counting and implication rules for market basket data. *ACM SIGMOD International Conference on Management of Data*, pages 255–264, 1997.

[38] R. A Brooks. A robust layered control system for mobile robot. *IEEE journal of robotics and automation*, RA-2 (1), Avril 1986.

[39] R Bruchez. *Optimiser sql server, dimensionnement, supervision, performances du moteur et du code sql.* Dunod, 2008.

[40] P Brunel, V Rollin, J Darmont, and L Gruenwald. Dbms auto-indexing using data mining techniques. Technical report, Université d'oklahoma USA et université Lumière Lyon II France, 2001.

[41] C Bucila, C Gehrke, J Kifer, and W. M White. Dualminer : A dual-pruning algorithm for itemsets with constraints. *Data Mining and Knowledge Discovery*, 7(3) :241–272, 2003.

[42] D Burdick, M Calimlim, and J Gehrke. Mafia : A maximal frequent itemset algorithm for transactional databases. . *International Conference on Data Engineering*, pages 443–452, 2001.

[43] P Busetta, R Ronnquist, A Hodgson, and A Lucas. Jack intelligent agents, components for intelligent agents in java. 1999.

[44] L. Cao. Integration of agents and data mining. Technical report, http ://www.staff.it.uts.edu.au/ lbcao/publication/publications.htm, 2005.

[45] L Cao, C Luo, and C Zhang. Agent mining interaction : an emerging area. *Lecture notes in computer science*, 4476 :60–73, 2007.

[46] L Cao, D Luo, Y Xiao, and Z Zheng. Agent collaboration for multiple trading strategy integration. *Proceedings of the 2nd KES international conference on agent and multi agent systems : technologies and applications, springer verlag*, pages 361–370, 2008.

[47] J. G Carbonell. Introduction : Paradigms for machine learning, in machine learning : Paradigms and methods. *J. G. Carbonell (Eds.), The MIT Press*, pages 1–9, 1990.

[48] C. Carpineto and G Romano. Galois : An order-theoretic approach to conceptual clustering. *Machine Learning Conference*, pages 33–40, 1993.

[49] B Chaib-draa, I Jarras, and B Moulin. Systèmes multi-agents : principes généraux et applications. In *Principes et architectures des systèmes multiagents*. Collection IC2, hermes science publication, 2002.

[50] D. D Chamberlin, M. M Astrahan, M. W Blasgen, J. N Gray, W. F King, B. G Lindsay, R Lorie, J. W Mehl, T. G Price, F Putzolu, P. G Selinger, M Schkolnick, D. R Slutz, I. L Traiger, B. W Wade, and R. A Yost. *A History and Evaluation of System R*. Magazine Communications of the ACM. Volume 24 Issue 10, 1981.

[51] C. Y Chan and Y. E Ioannidis. Bitmap index design and evaluation. *In proceedings of the ACM SIGMOD international conference on management of data*, pages 355–366, 1998.

[52] S Chaudhuri. Index selection for databases : A hardness study and a principled heuristic solution. *IEEE transactions on knowledge and data engineering*, 16(11) :1313–1323, 2004.

[53] S Chaudhuri and U Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1) :65–74, 1997.

[54] S Chaudhuri and V.R Narasayya. An efficient cost-driven index selection tool for microsoft sql server. *23rd international Conference on Very Large Data Bases (VLDB 97)*, pages 146–155, 1997.

[55] S Chaudhuri and V.R Narasayya. Autoadmin "what-if" index analysis utility. *International Conference on Management of Data, ACM SIGMOD*, pages 367–378, 1998.

[56] S Choenni, H.M Blanken, and T Chang. Index selection in relational databases. *5th International Conference on Computing and Information (ICCI 93)*, pages 491–496, 1993.

[57] C. H Choi, J. X Yu, and G Gou. What difference heuristics make : Maintenance-cost view-selection revisited. *3rd International Conference Web-Age Information Management*, pages 247–258, 2002.

[58] J Collis, D Ndumu, and S Thompson. Zeus methodology documentation, role modelling guide, three case studies : application realisation guide, and runtime guide. *http ://www.labs.bt.com/projects/agents/zeus*, 1999.

[59] B. A Davey and H. A Priestley. *Introduction to lattices and order.* Cambridge University Press, 1994.

[60] S Daviet. Modélisation des protocoles d'interaction d'une plateforme de simulation multi agents. *Mémoire de dea, université de nantes*, 2004.

[61] L De raedt and S Kramer. The levelwise version space algorithm and its application to molecular fragment finding. *International joint conference on artificial intelligence*, pages 853–862, 2001.

[62] Y Demazeau. From interactions to collective behaviour in agent-based systems. 1995.

[63] Y Demazeau and J-P Müller. Decentralized artificial intelligence. *Decentralized A.I. Elsevier science publishers*, 1 :3–13, 1990.

[64] R Derakhshan, F Dehne, O Korn, and B Stantic. Simulated annealing for materialized view selection in data warehousing environment. *In proceedings of the 24th IASTED international conference on database and applications*, pages 89–94, 2006.

[65] R Derakhshan, B Stantic, O Korn, and F Dehne. Parallel simulated annealing for materialized view selection in data warhousing environments. *International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2008)*, 2008.

[66] A Dogac, A.Y Erisik, and A Ikinci. An automated index selection tool for oracle7 : Maestro 7. *Technical Report LBNL/PUB-3161, Software Research and Development Center*, 1994.

[67] A Drogoul. *De la simulation multi agent à la résolution collective de problèmes.* PhD thesis, Université paris VI, 1993.

[68] A Drogoul. *De la simulation multi agent à la résolution collective de problèmes.* PhD thesis, Université paris VI, 1993.

[69] J. C Dunn. A fuzzy relative of the isodata process and its use in detecting compact well separated clusters. *cybernetics Journal*, 3 No 3 :32–57, 1973.

[70] E.H Durfee, V.R Lesser, and Corkill D.D. Cooperative distributed problem solving. *The handbook of artificial intelligence, Addison Wesley.*, IV :83–184, 1989.

[71] J Erceau and J Ferber. Intelligence artificielle distribuée. *La recherche*, 22(233) :750–758, 1991.

[72] U. M Fayyad, G Piatetsky-Shapiro, and P Smyth. Data mining and knowledge discovery in databases : introduction to the special issue. *Communications of the ACM*, 39, 1999.

[73] Y.A Feldman and J Reouven. A knowledge-based approach for index selection in relational databases. *Expert System with Applications*, 25(1) :15–37, 2003.

[74] J Ferber. *Objet et agents : une étude des structures de représentation et de communication en intelligence artificielle.* PhD thesis, Université de paris VI. France, 1989.

[75] J Ferber. Les systèmes multi-agents : vers une intelligence collective. *InterEditions, Paris*, 1995.

[76] J Ferber and A Drogoul. Using reactive multi agent systems in simulation and problem solving. *Distributed artificial intelligence : Theory and praxis, N. M. Avouris and L. Gasser (eds.), kluwer academic publishers, London*, 1992.

[77] J Ferber and M Ghallab. Problématiques des univers multi agents intelligents. *Journées nationales PRC-GRECO intelligence artificielle*, pages 295–320, 1988.

[78] I. A Ferguson. On supporting rational behavior in real-time multi-agent domains. *AAAI full symposium on rational agency : concepts, theories, models and applications, cambridge*, pages 61–65, November 1995.

[79] S Finkelstein, M Schkolnick, and P Tiberio. Physical database for relational databases. 1988.

[80] S.J Finkelstein, M Schkolnick, and P Tiberio. Dbdsgn : A physical database design tool for system r. *IEEE database eng, Bull*, 5(1) :9–11, 1982.

[81] FIPA. Fipa interaction protocol specifications. Technical report, Foundation for intelligent physical agents, 2002.

[82] tc communication Fipa. Fipa contract net interaction protocol specification. Technical report, Foundation for intelligent physical agents, 03/12/2002.

[83] L. N Foner. What's an agent, anyway ? : A sociological case study. Technical report, MIT media lab, 1993.

[84] Foundation for intelligent physical agents. Fipa acl message structure specification. *http ://www.fipa.org*, 2003.

[85] M.R Frank, E Omiecinski, and S.B Navathe. Adaptive and automated index selection in rdbms. *3rd international conference on extending database technology (LNCS EDBT 92)*, 580 :277–292, 1992.

[86] W.J Frawley, G Piatetsky-Shapiro, and C Matheus. Knowledge discovery in databases : An overview. *In Knowledge Discovery In Databases, eds. G. Piatetsky-Shapiro, and W. J. Frawley, AAAI Press/MIT Press, Cambridge, MA*, pages 1–30, 1991.

[87] C.H Fung, K Karlapalem, and Q Li. Cost driven vertical class partitioning for methods in object oriented databases. *VLDB Journal*, 12 (3) :187–210, 2003.

[88] le mahec Gaël. Utilisation des arbres de radicaux pour les algorithmes de datamining sur grille de calcul. *DEA en Informatique parallèle répartie et combinatoire, université de picardie, france*, 2004.

[89] J.R Galliers. Modeling autonomous belief revision in dialogue. *In yves demazeau et pierre müller (éd.) decentralized artificial intelligence 2 : Proceedings of the second european workshop on autonomous agents in a multi agents world, elsevier science*, 1991.

[90] B Ganter and R Willer. Formal concept analysis : mathematical foundations. *Lecture notes in Artificial Intelligence, Springer-Verlag*, 1257 :332–341, 1999.

[91] G Gardarin, P Pucheral, and F Wu. Bitmap based algorithms for mining association rules. *Actes des 14 émes journées bases de données avancées*, pages 157–175, 1998.

[92] M Garofalakis, R Rastogi, and K Shim. Spirit : Sequential pattern mining with regular expression constraints. *In VLDB'99, Edinburgh, UK, Morgan Kaufmann*, pages 223–234, 1999.

[93] L Getoor, B Taskar, and D Koller. Selectivity estimation using probabilistic models, sigmod 01. pages 461–472, 2001.

[94] R Godin. Complexité de structures de treillis. *Annales des sciences mathématiques du québec*, 13(1) :19–38, 1989.

[95] J Goldstein and P Larson. Optimizing queries using materialized views : A practical, scalable solution. *In proceedings of the ACM SIGMOD*, pages 259–269, 2001.

[96] M Golfarelli, S Rizzi, and E Saltarelli. Index selection for data warehousing. *4th International Workshop on Design and Management of Data Warehouses (DMDW 02)*, pages 33–42, 2002.

[97] K Gouda and M. J Zaki. Genmax : an efficient algorithm for mining maximal frequent itemsets. *Data Mining and Knowledge Discovery : An International Journal*, 11 :1–20, 2005.

[98] J Gray, A Bosworth, A Layman, and H Pirahesh. Datacube : A relational operator generalizing group by, cross-tab and sub-total. *proceeding of the 12 international conference on data engineering*, pages 152–159, 1996.

[99] S Grumbach, M Rafanelli, and L Tininini. On the equivalence and rewriting of aggregate queries. *Acta informatica*, 40(8) :529–584, 2004.

[100] J. R Gruser. *Modèle de coût pour l'optimisation de requête objet.* PhD thesis, Université de Paris VI, 1996.

[101] Z Guessoum and al. Monitoring and organizational-level adaptation of multi agent systems. *Proceedings of the third international joint conference on autonomous agents and multi agent systems*, 2, 2004.

[102] Z Guessoum and al. Adaptive replication of large-scale multi agent systems : towards a fault-tolerant multi-agent platform. *Proceedings of the fourth international workshop on Software engineering for largescale multi-agent systems. ACM SIGSOFT software engineering notes*, 30(4), 2005.

[103] Z Guessoum and M Occello. *Environnements de développement In "principes et architecture des systèmes multi-agents".* Collection IC2, hermès science publications, 2001.

[104] S Guha, R Rastogi, and K Shim. Cure : an efficient clustering algorithm for large databases. *In proceedings of the international conference on management of data*, Number 2, Vol 27 :73–84, 1998.

[105] T.I Gundem. Near optimal multiple choice index selection for relational databases. *Computers & Mathematics with Applications*, 37(2) :111–120, 1999.

[106] H Gupta. Selection of views to materialize in a data warehouse. *In proceeding of 6th International conference database theory (ICDT)*, pages 98–112, 1997.

[107] H Gupta, V Harinarayan, A Rajaraman, and J.D Ullman. Index selection for olap. *In Proceeding of 13th International conference on data engineering (ICDE)*, pages 208–219, 1997.

[108] H Gupta and S Mumick. Selection of views to materialize under a maintenance cost constraint. *in proceeding of 7th international conference on extended database theory*, pages 453–470, 1999.

[109] H Gupta and S Mumick. Selection of views to materialize in a data warehouse. *IEEE transactions on knowledge and data engineering*, 17(11) :24–43, 2005.

[110] O Gutknecht, J Ferber, and F Michel. Madkit : une plate-forme multi agents générique. *Laboratoire d'informatique robotique et micro électronique de montpellier*, 2000.

[111] O Gutknecht, J Ferber, and F Micheln. From agents to organizations : an organizational view of multi agent systems. *Agent oriented software engineering*, 2935 :214–230, 2004.

[112] R. W Hamming. Error detecting and error correcting codes. *Bell system technical journal*, 29(2) :147–160, 1950.

[113] E-H Han, G Karypis, V Kumar, and B Mobasher. Clustering based on association rule hypergraphs. *In research issues on data minnig and knowledge discovery*, 1997.

[114] J Han and M Kamber. Data mining : concepts and techniques. *Morgan kaufmann*, 2001.

[115] J Han, J Pei, and Y Yin. Mining frequent patterns without candidate generation. *In proceedings of the ACM SIGMOD Conference*, pages 1–12, 2000.

[116] V Harinarayan, A Rajaraman, and J.D Ullman. Implementing data cubes efficiently. *In proceedings of international conference of ACM SIGMOD*, pages 205–216, 1996.

[117] B Hayes-Roth. A blackboard model of control hpp83-38. Technical report, Stanford university, 1984.

[118] J Hipp, U Guntzer, and G Nakhaeizadeh. Algorithms for association rule mining a general survey and comparison. *SIGKDD explorations*, 2(1) :58–64., 2000.

[119] M. N Huhns. Distributed artificial intelligence. *Morgan Kaufman*, 1987.

[120] C. A Hurtado, 0. A Mendelzon, and A Vaisman. A maintaining data cubes under dimension updates. *Proceedings of the international conférence on data engineering*, pages 346–355, 1999.

[121] W.H Inmon. Building the data warehouse. *John Wiley*, 1992.

[122] IS-Net. Les agrégats. Technical report, Haute ecole spécialisé de suisse occidentale, 2002.

[123] A. K Jain, M. N Murty, and P. J Flynn. Data clustering : a review. *ACM computing surveys*, 31, issue 3 :263–323, 1999.

[124] N.R Jennings, K Sycara, and M.J Wooldridge. A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1 :275–306, 1998.

[125] B Jeudy. *Optimisation de requêtes inductives : application à l'extraction sous contraintes de règles d'association.* PhD thesis, Institut national des sciences appliquées (INSA) de Lyon. France, 2002.

[126] T Johnson. Performance measurements of compressed bitmap indices. *In Very Large Data Base*, 278-289, 1999.

[127] P Jouve and N Nicoloyannis. Kerouac, an algorithm for clustering categorical data sets with practical advantages. *International workshop on data mining for actionable knowledge*, 2003.

[128] P Kalnis, N Mamoulis, and D Papadias. View selection using randomized search. *Data and knowledge engineering*, 42(1) :89–111, 2002.

[129] H.J Karloff and M Mihail. On the complexity of the view-selection problem. *In proceedings of PODS*, pages 167–173, 1999.

[130] D Kayser, A.M Florea, and S.G Pentiuc. *Intelligence artificielle et agents intelligents.* 2004.

[131] R Kimball. *A dimensional modeling manifesto.* DBMS magazine, August, 1997.

[132] D Kinny, J Treur, L Gasser, S Clark, and J Müller. Intelligent agents iv agent theories, architectures, and languages. *Lecture notes in computer science*, 1365, 1998.

[133] M Klemettinen, H Mannila, P Ronkainen, H Toivonen, and A.I Verkamo. Finding interesting rules from large sets of discovered association rules. *In third international conference on Information and Knowledge Management*, pages 401–407, 1994.

[134] Y Kotidis and N Roussopoulos. Dynamat : A dynamic view management system for data warehouses. *Proc of the ACM SIGMOD*, pages 371–382, 1999.

[135] J Kratica, I Ljubic, and D Tosic. A genetic algorithm for the index selection problem : Applications of evolutionary computing. *EvoWorkshops LNCS*, 2611 :281–291, 2003.

[136] T Kumar, V and S Kumar. Materialized view selection using simulated annealing. *Lecture notes in computer science*, 7678 :168–179, 2012.

[137] S Labidi and W Lejouad. De l'intelligence artificielle distribuée aux systèemes multi-agents. *Rapport de recherche*, 1993.

[138] W Labio, D Quass, and B Adelberg. Physical database design for data warehouses. *Proceedings of the international conference on data engineering (ICDE)*, pages 277–288, 1997.

[139] S Lallich and O Teytaud. Evaluation et validation de l'intérêt des règles d'association. *Revue des nouvelles technologies de l'information*, pages 193–217, 2004.

[140] M Lee and J Hammer. Speeding up materialized view selection in data warehouses using a randomized algorithm. *International journal of cooperative information system*, vol 10, no 3 :327–353, 2001.

[141] C Lenay. Coopération et intentionnalité. *Quatrième journée francophones sur l'intelligence artificielle distribuée et système multi agents*, 265-272, 1996.

[142] B Lent, A. N Swami, and J Widom. Clustering association rules. *In proceeding of the thirteen international conference of data engineering*, pages 220–231, 1997.

[143] W Liang, H Wang, and M.E Orlowska. Materialized view selection under the maintenance time constraint. *Data and Knowledge Engineering*, pages 203–216, 2001.

[144] J Lind. Iterative software engineering for multi agent systems, the massive method. *Lecture notes in artificial intelligence, springer verlag, .*, 1994, 2001.

[145] L. G Liu. Introduction to combinatorial mathematics. *McGraw Hill*, 1968.

[146] Lumigent. Logexplorer. *http : //www.lumigent.com/logexplorer*, 2000.

[147] J Maassen, R.V Nieuwpoort, R Veldema, H Bal, T Kielmann, C Jacobs, and R Hofman. An efficient implementation of java remote method invocation. *Proceedings of the seventh acm sigplan symposium on principles and practice of parallel programming*, pages 173–182, 1999.

[148] J. B MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5 th berkeley symposium on mathematical statistics and probability, berkeley, university of california press*, 1 :281–297, 1967.

[149] A McCallum, K Nigam, and L Ungar. Efficient clustering of high dimensional data sets with application to reference matching. *In sixth ACM SIGKDD International conference on knowledge discovery and data Mining*, 2000.

[150] Sun microsystems. Rpc, remote procedure call protocol specification. 1988.

[151] J. P Müller and M Pischel. Modelling reactive behaviour in vertically layered agent architectures. *The eleventh european conference on artificial intelligence*, pages 709–713, 1994.

[152] H Naacke. *Modèles de coût pour médiateurs de bases de données hétérogènes.* PhD thesis, Université de versailles saint-quentin en yvelines, France, 1999.

[153] T. P Nadeau and T. J Teorey. Achieving scalability in olap materialized view selection. *in 5th acm international workshop on data warehousing and olap*, pages 28–34, 2002.

[154] S Navathe and M Ra. Vertical partitioning for database design a graphical algorithm. *acm sigmod*, pages 440–450, 1989.

[155] H Necir. A data mining approach for efficient selection bitmap join index. *International journal of data mining modelling and management*, Number 3, Vol 2 :238–251, 2010.

[156] H Necir. Efficient data mining strategy in index selection problem with convertible constraint. *In IEEE international conference on system man and cybernetics, edited by LNCS*, 2010.

[157] H Necir, L Bellatreche, and R Missaoui. Dynaclose une approche de fouille de données pour la sélection des index de jointure binaires dans les entrepôts de données. *Actes des 3èmes journées francophones sur les entrepôts de données et l'analyse en ligne, revue des nouvelles technologies d'information*, 3 :83–98, 2007.

[158] H Necir and H Drias. A distributed clustering with intelligent multi agents system for materialized views selection. *In proceedings of the 4rth international conference : sciences of electronic, technologies of information and telecommunications, published in advances in intelligent and soft Computing, springer verlag, berlin, germany*, pages 417–426, 2012.

[159] H Necir, H Drias, and y Mechedou. Approche data mining pour la sélection des vues matérialisées. *7iéme manifestation des jeunes chercheurs en sciences et technologies de l'information et de la communication, avignon*, 2009.

[160] H Nwana. Software agents : An overview. *The knowledge engineering review*, 11 (3) :205–244, 1996.

[161] council O. Apb1 olap benchmark. *release http ://www.olapcouncil.org/ research/ resrchly.htm*, 1998.

[162] J Odell, H.V.D Parunak, and B Bauer. Extending uml for agents. *Proceedings of the agent oriented information systems workshop at the 17th national conference on artificial intelligence, springer*, pages 3–17, 2000.

[163] J. Odell, H.V.D Parunak, and B Bauer. Representing agent interaction protocols in uml. *In proceedings on the first international workshop on agent oriented software engineering*, pages 121–140, 2000.

[164] OMG. The common object request broker, architecture and specification revision 2.2. *omg document*, 1998.

[165] P O'Neil and D Quass. Improved query performance with variant indexes. *In proceedings of the ACM SIGMOD international conference on management of data (SIGMOD 1997)*, pages 38–49, 1997.

[166] Oracle. Oracle database performance tuning guide. *10g Release 2 (10.2), part number B14211-01*, 2001.

[167] Oracle. Guide d'entreposage de données oracle. *Rapport technique, la bibliothèque de documentation oracle 11g*, 2007.

[168] K Parsaye. Surveying decision support : new realms of analysis information discovery. *Inc*, 1996.

[169] H.V.D Parunak. Manufacturing experience with the contract net. *Morgan kaufmann*, pages 285–310, 1987.

[170] N Pasquier, Y Bastide, R Taouil, and L Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems, Elsevier Science*, 24(1) :25–46, 1999.

[171] N Pasquier, Y Bastide, R Touil, G Stumme, and L. Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. *Lecture notes in computer science*, 1861 :972–986, 2000.

[172] N Pasquier, Bastide Y, Taouil R, and Lakhal L. Discovering frequent closed itemsets. *Proceedings of 7th international conference on database theory (ICDT99), LNCS Springer Verlag*, 1540 :398–416, 1999.

[173] J Pei, J Han, and L. V. S Lakshmanan. Pushing convertible constraints in frequent itemset mining. *An international journal of data mining and knowledge discovery*, 8(3), Kluwer Academic Publishers :227–252, 2004.

[174] J Pei, J Han, and R Mao. Closet an efficient algorithm for mining frequent closed itemsets. *Proceedings of the ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, 2000.

[175] J.C Perez. De nouvelles voies vers l'intelligence artificielle. 1988.

[176] G Picard. *Méthodologie de développement de systèmes multi agents adaptatifs et conception de logiciels à fonctionnalité émergente.* PhD thesis, Université Paul Sabatier de Toulouse III. France, 2004.

[177] R Ramakrishman. Database management systems. *WCB/mcgraw hill*, 1998.

[178] J Rao, C Zhang, N Megiddo, and G.M Lohman. Automating physical database design in a parallel database. *SIGMOD 02*, pages 558–569, 2002.

[179] A Rauber, E Pampalk, and J Paralic. Empirical evaluation of clustering algorithms. *Journal of information and organizational sciences*, 24(2) :195–209, 2000.

[180] P Rigaux. Support de cours de bases de données. 2002.

[181] S Rizzi and E Saltarelli. View materialization vs indexing : balancing space constraints in data warehouse design. *In the 15th international conference on advanced information systems engineering*, pages 502–519, 2003.

[182] S Russel and P Norvig. *Artificial intelligence, a modern approach.* 1995.

[183] S. J Russell and P Norvig. *Artificial intelligence : A modern approach.* Prentice-Hall, 1995.

[184] R Rymon. Search through systematic set enumeration. *International conference on principles of knowledge representation and reasoning*, pages 539–550, 1992.

[185] A Salleb-Aouissi. *Recherche de motifs fréquents pour l'extraction de règles d'association et de caractérisation.* PhD thesis, Université d'orléans. France, 2003.

[186] A Sanjay, V.R Narasayya, and B Yang. Integrating vertical and horizontal partitioning into automated physical database design. *Proceedings of the acm sigmod international conference on management of data*, pages 359–370, 2004.

[187] A Sanjay, C Surajit, and V. R Narasayya. Automated selection of materialized views and indexes in microsoft sql server. *In international conference on Very Large Data Bases (VLDB 2000)*, 2000.

[188] A Savasere, E Omiencinski, and S Navathe. An efficient algorithms for mining association rules in large databases. *21st international conference on very large data bases*, pages 432–444, 1995.

[189] J Searle. *Speech acts, an essay in the philosophy of language.* Cambridge university Press, london, 1969.

[190] Y Shoham. Agent oriented programming. *Artificial intelligence*, 60(1) :51–92, 1993.

[191] A Shukla, P Deshpande, and J Naughton. Materialized view selection of multidimensional datasets. *In proceeding of the 24th international conference on very large data bases (VLDB)*, pages 488–499, 1998.

[192] R.G Smith. The contract net protocol, high level communication and control in a distributed problem solver. *Ieee transactions on computers*, 29(12) :1104–1113, 1981.

[193] R Srikant, Q Vu, and R Agrawal. Mining association rules with item constraints. *International conference on Knowledge Discovery and Data Mining, AAAI Press*, pages 67–73, 1997.

[194] L Steels. Cooperation between distributed agents through self organisation. *Decentralized AI, eds Y Demazeau and J-P Müller. Elsevier Science Publishers B.V*, pages 175–196, 1990.

[195] M Steinbrunn, J Moerkotte, and A Kemper. Heuristic and randomized optimization for the join ordering problem. *International journal on very large data base*, 6 :191–208, 1997.

[196] G Stumme, R Taouil, Y Bastide, N Pasquier, and L Lakhal. Intelligent structuring and reducing of association rules with formal concept analysis. *Proceeding on Artificial Intelligence conference (KI 2001). LNAI 2174. Springer-verlag*, pages 335–350, 2001.

[197] R. B Systems. Star schema processing for complex queries. Technical report, White Paper, 1997.

[198] Reticular systems inc. Agentbuilder user guide. *http ://agentbuilder.com*, 1999.

[199] M Theodoratos and T Sellis. Data warehouse configuration. *In the proceedings of the international conference on very large databases*, pages 126–135, 1997.

[200] H Thomas, A Datta, and I Viguier. A conceptual model and algebra for on-line analytical processing in decision support databases. *Research report, university of arizona*, 1997.

[201] S. R Thomas. *PLACA : an agent oriented programming language.* PhD thesis, Stanford university, 1993.

[202] H Toivonen. Sampling large databases for association rules. *Proceedings of the twenty fourth international conference on very large data bases*, pages 134–145., 1996.

[203] J. D Ullman. Efficient implementation of data cubes via materialized views. *In the 2nd international conference on knowledge discovery and data mining (KDD 96)*, pages 386–388, 1996.

[204] P Valduriez. Join indices. *ACM transaction on database systems*, 12(2) :218–246, 1987.

[205] G Valentin, Zuliani M, Zilio D, Lohman G, and Skelley A. Db2 advisor : An optimizer smart enough to recommend its own indexes. *16th international conference on data engineering*, pages 101–110, 2000.

[206] W Vickrey. Counterspeculation, auctions and competitve sealed tenders. *Journal of finance*, 1961.

[207] J Vàzquez-Salceda, V Dignum, and F Dignum. Organizing multi agent systems. Technical report, Institute of information and computing sciences, utrecht university„ 2004.

[208] A Wang, R Conradi, and C Liu. A multi-agent architecture for cooperative software engineering. *Twelfth international conference of software engineering and knowledge engineering*, 2000.

[209] J Wang, J Han, and J Pei. Closet+ : searching for the best strategies for mining frequent closed itemsets. *Ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003.

[210] G Weib and S Sen. Adaption and learning in multi agent systems. *Lecture notes in computer science 1042, Springer*, 1995.

[211] G Weiss. *Multiagent systems : a modern approach to distributed artificial intelligence.* MIT press, 1999.

[212] D Weyns, H Van, D Parunak, F Michel, T Holvoet, and J Ferber. Environments for multi agent systems state of the art and research challenges. *In proceeding of first international workshop environments for multi agent systems*, pages 1–47, 2004.

[213] K Whang. Index selection in relational databases. *International conference on foundations of data organization (FODO 85)*, pages 487–500, 1987.

[214] J Widom. Research problems in data warehouse. *In 4th international conference on information, knowledge and managment*, pages 25–30, 1995.

[215] R Wille. Restructuring lattice theory : An approach based on hierarchies of concepts, ordered sets. *Ivan Rival Ed, NATO advanced study institute*, pages 445–470, 1981.

[216] R Wille. Concept lattices and conceptual knowledge systems. *Computers and Mathematics with Applications*, vol. 23, n°6-9 :493–515, 1992.

[217] K. Y. M Wong, S. W Lim, and P Luo. Diversity and adaptation in large population games. *International journal mod phys*, 18 :222–243, 2004.

[218] M Wooldridge. On the sources of complexity in agent design. *Applied artificial intelligence*, 14(7) :623–644, 2000.

[219] M Wooldridge. *An introduction to multi agent systems.* John Wiley & Sons, 2002.

[220] M Wooldridge and NR Jennings. Intelligent agents : theory and practice. *The knowledge engineering review*, 10(2) :115–152, 1995.

[221] M Wooldridge, N.R Jennings, and D Kenny. The gaia methodology for agent oriented analysis and design. *Journal of autonomous agents and MAS*, 2000.

[222] R Wrembel. Data warehouse performance, selected techniques and data structures. *Lecture Notes in Business Information Processing*, 96 :27–62, 2012.

[223] K Wu, E Otoo, and A Shoshani. A performance comparison of bitmap indexes. *In proceedings of the tenth international conference on information and knowledge management*, pages 559–561, 2001.

[224] K Wu, E Otoo, and A Shoshani. On the performance of bitmap indices for high cardinality attributes. *Proceedings of the thirtieth international conference on Very large data bases*, 30 :24–35, 2004.

[225] M Wu. Query optimization for selections using bitmap. *in ACM SIGMOD international conference on management of data*, pages 227–238, 1999.

[226] M. C Wu and A Buchmann. Encoded bitmap indexing for data warehouses. *Proceedings of the fourteenth international conference on data engineering*, pages 220–230, 1998.

[227] R Xu and D Wunsch. Survey of clustering algorithms. *ieee transactions on neural networks*, 16(3) :645–678, 2005.

[228] J Yang, K Karlapalem, and Q Li. Algorithm for materialized view design in data warehousing environment. *23rd International Conference on Very Large Data Bases*, pages 136–145, 1997.

[229] J.X Yu, X Yao, C-H Choi, and G Gou. Materialized view selection as constrained evolutionary optimization. *IEEE transactions on systems, man, and cybernetics - Part C : Applications and Reviews*, 33, no, 4 :458–467, 2003.

[230] M. J Zaki. Generating non redundant association rules. *Proceedings of the 6th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 34–43, 2000.

[231] M. J Zaki and C. J Hsiao. Charm : an efficient algorithm for closed itemset mining. *2nd SIAM international conference on data mining (ICDM02)*, pages 457–473., 2002.

[232] M.J Zaki and M Ogihara. Theoretical foundations of association rules. *Proceding of 3rd SIGMOD worksh, research Issues in data mining and knowledge discovery*, pages 1–8, 1998.

[233] C Zhang and J Yang. Genetic algorithm for materialized view selection in data warehouse environments. *International conference on data warehousing and knowledge discovery (DaWaK), LNCS Springer.*, vol 1676 :116–126, 1999.

[234] C Zhang, X Yao, and J Yang. An evolutionary approach to materialized view selection in a data warehouse environment. *IEEE transactions on systems, man, and cybernetics - Part C : Applications and Reviews*, vol 31, no 3 :282–294, 2001.

[235] D Zighed and R Rakotomalala. Data mining. *Editions techniques de l'ingénieur*, H3 744 :1–26, 2002.