N° D'ORDRE : 10/2009-M/M

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET

DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE

« HOUARI BOUMEDIENE »

FACULTE DE MATHEMATIQUES

MEMOIRE

Présenté pour l'obtention du diplôme de MAGISTER

EN : MATHEMATIQUES

Spécialité : Recherche Opérationnelle : Génie Mathématiques

Par : ZEBDI Belkacem

Sujet :

Matérialisation de vues dans un entrepôt de données basée sur la densité

Soutenu publiquement le 19/11/2009, devant le jury composé de :

Mr- M. BOUDHAR	*Maître de conférences/A à l'U.S.T.H.B*	*Président*
Mr- S. BOUROUBI	*Maître de conférences/A à l'U.S.T.H.B*	*Directeur de mémoire*
Mr- A. BERRACHEDI	*Professeur à l'U.S.T.H.B*	*Examinateur*
Mr- A. BOUKRA	*Maître de conférences/A à l'U.S.T.H.B*	*Examinateur*

Dedicaces

A la mémoire de ma mére, qui est la clé de ma réussite de mes études, son rêve de voir son fils dans un grade d'études très avancées m'a toujours poussé de faire des efforts pour réaliser son souhait.

A mon pére, qui m'aide chaque fois que j'en ai besoin de lui, et je prie dieu qu'il le guérisse de sa maladie.

Je dédie cette thèse a mes quatre soeurs **RABIA**, **SAMIA**, **ASSIA**, **MERIEM**, qui essayent de me conforter chaque fois où je passe des périodes pénibles.

A mon frère **AMINE**, à mes deux frères **MOHAMED** et **BILLEL**, qui j'aime bien et je prie dieu pour qu'il les aide à surpasser leurs handicap.

A ma grande mère **KHADOUDJA**, à ma tente **HOURIA** et son mari **RABAH**, à mon oncle **ABDERRAHMANE**, et à toute ma grande famille.

A tous mes collégues de l'université et les fidèles amis de mon quartier.

A tous ceux qui aiment **KACI** et à tous ceux que **KACI** aime.

Remerciements

Je voudrais remercier profondément Monsieur **Sadek BOUROUBI** mon directeur de thèse (Maître de conférences à la Faculté de Mathématiques, U.S.T.H.B.) de m'avoir proposé ce sujet et de m'avoir encadré pendant toutes ces années. Je le remercie vivement pour son aide, ses conseils, sa patience, sa disponibilité, ses orientations et ses indications. Mes reconnaissances pour lui de m'avoir fait confiance et de m'avoir mis à l'aise tout au long de l'avancement de cette thése.

Ma reconnaissance va ensuite aux personnes qui me font l'honneur de composer ce jury : Monsieur **Mourad BOUDHAR**, Maître de conférences à la Faculté de Mathématiques, U.S.T.H.B., pour avoir accepté de présider ce jury.

Monsieur **Abdelhafid BERRACHEDI**, Professeur à la Faculté de Mathématiques, U.S.T.H.B., Monsieur **Abdelmajid BOUKRA** (Maître de conférences à l'institut d'informatique et d'électronique, U.S.T.H.B.) d'avoir accepté d'être les examinateurs de ce travail.

Je remercie encore Monsieur **Abdelmajid BOUKRA** pour ses conseils et ces explications, pour le temps qui m'a consacré et pour sa présence lors de mes exposés.

Je remercie Monsieur **Kamel BOUKHETALA** (Professeur à la Faculté de Mathématiques, U.S.T.H.B.) mon directeur de mémoire d'ingéniorat, Année 2005, pour m'avoir permis de soutenir avant le concours de PG pour le préparer. De même je remercie encore Monsieur **Abdelhafid BERRACHEDI** pour d'avoir faciliter les inscriptions pour le concours afin de permettre à tous les étudiants de le préparer aisement.

Je remercie mes amis **Mustapha CHOUIHA**, **Fayçal GOUGAM**, **Noureddine BOURRAS** et **Amine BENACER** et l'ami de mon pére **Mohamed** pour leurs aides.

Je remercie les étudiants de PG Statistiques qui m'ont aidé à rédiger ce mémoire.

Résumé

Aujourd'hui les entrepôts de données jouent un rôle important dans le domaine d'informatique décisionnelle. L'un des paramètres de performance est le temps de réponse des reqûetes de type OLAP. La matérialisation de vues est parmi les techniques utilisées pour réduire ce temps de réponse.

Nous adressons dans ce mémoire, le problème de matérialisation de vues sous contrainte de maintenance. Ce problème consiste à sélectionner l'ensemble de vues à matérialiser sous contrainte de maintenance, en minimisant le coût de réponse total.

Nous proposons deux algorithmes pour résoudre ce problème, le premier est basé sur le principe des colonies de fourmis et l'autre sur la densité. Les résultats expérimentaux ont montré l'efficacité de ces deux algorithmes par rapport aux algorithmes existants.

Mots clés : Entrepôt de données, Technologie OLAP, Matérialisation de vues, Algorithme de colonies de fourmis, Notion de densité.

Abstract

Today Data warehouses play an important role in the field of decision support. One of the parameters of performance is the time to answer OLAP requests. The materialization of views is among the techniques used to reduce the response time.

We address in this thesis, the problem of materialized views under maintenance constraint. This problem is to select the set of views to materialize under the maintenance cost constraint, minimizing the total cost of reponse.

We propose two algorithms to solve this problem, the first is based on the principle of ant colony and the second on the density. The experimental results showed the effectiveness of our algorithms compared to existing algorithms.

Keywords : Data Warehouse, OLAP Technology, Views materialization, Ant Colony algorithme, Density notion.

Table des matières

Table des figures

Introduction Générale

L'idée d'entrepôt de données a été introduite afin de prendre des décisions efficaces. L'un des paramètres déterminant dans cette efficacité est le temps de réponse des requêtes.

L'une des techniques utilisées pour réduire ce temps de réponse est de sélectionner un ensemble de vues à matérialiser.

Le problème est donc de trouver l'ensemble optimal de vues qui minimise le temps de réponse total, en réspectant certaines contraintes de ressources .

Les algorithmes exactes de la recherche opérationnelle ne donnent de bons résultats que pour des problèmes de petites tailles. Pour les problémes de tailles plus importantes, des méta-heuristiques sont utilisées pour donner des solutions proches de la solution optimale.

Dans ce travail, nous exploitons l'algorithme de colonie de fourmis pour résoudre le probléme de la matérialisation de vues sous contrainte de la maintenance, en plus nous allons voir la matérialisation de vues dans un entrepôt de données basé sur la densité.

Ce mèmoire est développé en cinq chapitres :

- Le premier chapitre est consacré aux définitions et à la terminologie utilisées dans les entrepôts de données et l'informatique décisionnelle.
- Aprés, on va voir comment résoudre le probléme de matérialisation de vues dans un entrepôt de données sous la contrainte de maintenance dans le deuxiéme chapitre.
- Le chapitre suivant exploite l'algorithme de colonies de fourmis pour la résolution du problème et compare l'algorithme Hybride avec un nouveau algorithme nommé **CFGC**.
- L'avant dernier chapitre traite la matérialisation de vues dans un entrepôt de données basé sur la densité, et l'intervention de la contrainte de maintenance.

– Enfin, dans le dernier chapitre, nous allons voir quelques résultats expérimentaux issues de l'application des algorithmes de matérialisation de vues traités dans cette thése ainsi qu'un exemple pratique.

Chapitre 1

Les entrepôts de données

1.1 Introduction

Aujourd'hui avec la mondialisation des marchés et des économies l'entreprise évolue dans un environnement de plus en plus vaste, de plus en plus concurrentiel et doit faire face à des risques et des contraintes toujours plus grands. L'entreprise collecte en permanence de nouvelles données, toujours en quête de nouvelles études et analyses. Toute cette masse de données est produite quotidiennement que ce soit par l'entreprise elle même ou par son environnement.
Pour aider et assister les décideurs dans leur processus de prise de décision, l'entrepôt de données constitue de nos jours un élément incontournable. L'objectif d'un ED est d'assembler toutes les parties qui constituent le système d'information pour former ainsi une seule structure de données plus facile à manipuler, à exploiter et à gérer.

1.2 Le décisionnel

Les sociétés de téléphone gardent au moins un an les positions géographiques et les consommations de leurs abonnés « mobiles ». Les grands magasins et les entreprises de vente par correspondance **(VPC)** conservent les achats de leurs clients (tickets de caisse en grande distribution, commandes en VPC), collectent des informations sur leurs clients grâce à des systèmes de cartes de fidélité ou de crédit, et achètent des bases de don-

nées géographiques et démographiques. Les sites web conservent des traces de connexions sur leurs sites marchands. En résumé, les entreprises dans un secteur très concurrentiel conservent les données de leur activité et achètent même des données.
Les motifs qui ont présidé à la conservation de ces données étaient : des obligations légales pour pouvoir justifier les facturations, des raisons de sécurité pour pouvoir détecter les fraudes, des motifs commerciaux pour suivre l'évolution des clients et des marchés. Quelle que soit la raison initiale, les entreprises se sont rendues compte que ces données pouvaient être une source d'informations à leur service. Ce constat, valable pour les sociétés du secteur marchand, peut être étendu à de nombreux domaines comme la médecine, la pharmacologie. Il faut donc définir des environnements permettant de mémoriser de grands jeux de données et d'en extraire de l'information.
Les structures qui accueillent ce flot important de données sont des entrepôts de données ou data warehouse. Ils sont construits sur une nouvelle architecture permettant d'extraire l'information, architecture bien différente de celle prévue pour l'informatique de production, basée elle sur des systèmes de gestion de bases de données relationnelles et des serveurs transactionnels. Un entrepôt de données est construit en l'alimentant via les serveurs transactionnels de façon bien choisie et réfléchie pour permettre aux procédures d'extraction de connaissances de bien fonctionner. L'organisation logique des données est particulièrement conçue pour autoriser des recherches complexes. Le matériel est évidemment adapté à cette utilisation.

1.3 L'informatique décisionnelle et l'informatique de production

Une des principales caractéristiques des systèmes de production est une activité constante constituée de modifications et d'interrogations fréquentes des données par de nombreux utilisateurs : ajouter une commande, modifier une adresse de livraison, rechercher les coordonnées d'un client, ... Il faut conserver la cohérence des données (il faut interdire la modification simultanée d'une même donnée par deux utilisateurs différents). Il s'agit

donc de privilégier un enregistrement rapide et sûr des données. L'inverse, les utilisateurs des systèmes d'information de décision n'ont aucun besoin de modification ou d'enregistrement de nouvelles données. Ils vont interroger le système d'information et les questions posées seront de la forme : « quelles sont les ventes du produit X pendant le trimestre A de l'année B dans la région C ». Une telle interrogation peut nécessiter des temps de calcul importants. Or, l'activité d'un serveur transactionnel ne peut être interrompue. Il faut donc prévoir une nouvelle organisation qui permette de mémoriser de grands jeux de données et qui facilite la recherche d'informations.
L'existence d'un entrepôt simplifiera cette tâche et permettra donc d'optimiser le temps de développement d'un projet d'extraction de connaissances, donc il est beaucoup plus simple de trouver une information pertinente dans une structure organisée pour la recherche de connaissance.

1.3.1 La production

Le modèle Entité-Association (ou modèle EA) est l'un des formalismes les plus utilisés pour la représentation conceptuelle des systèmes d'information. Il permet de construire des modèles de données dans lesquels on cherche à tout prix à éviter des redondances. Toute donnée mémorisée plus d'une fois est source d'erreurs, d'incohérences, et va pénaliser les temps d'exécution et complexifier les procédures d'ajout, de suppression ou de modification.
D'un point de vue logique, ce sont les bases de données relationnelles qui ont su au mieux représenter ces modèles conceptuels. Les éditeurs de logiciels et les architectes d'ordinateurs ont dû fournir un effort important pour les rendre efficaces. Aujourd'hui les bases relationnelles ont acquis une maturité satisfaisante pour être largement utilisées et diffusées dans de nombreuses organisations, à toute échelle.
Conserver la cohérence de la base de données, c'est l'objectif et la difficulté principale pour l'informatique de production. Les systèmes transactionnels (temps réel) **OLTP** (*On-Line Transaction Processing*) garantissent l'intégrité des données. Les utilisateurs accèdent à des éléments de la base par de très courtes transactions indécomposables, isolées. Ils y accèdent très souvent pour des opérations d'ajout, suppression, modification mais aussi

de lecture. L'isolation permet de garantir que la transaction ne sera pas perturbée ni interrompue. Les contrôles effectués sont élémentaires. La brièveté garantit que les temps de réponse seront acceptables (inférieurs à la seconde) dans un environnement avec de nombreux utilisateurs.

Bien sûr, dans le souci de la garantie des performances, une requête dont le calcul prendrait trop de temps est inacceptable. Par exemple, une jointure sur de grosses tables à l'aide de champs non indexés est interdite. Les travaux sur une telle base sont souvent simples et répétitifs. Dès lors qu'un travail plus important est nécessaire, l'intervention de programmeurs et d'administrateurs de la base est requise et une procédure ad-hoc est créée et optimisée.

Le modèle Entité-Association et sa réalisation dans un schéma relationnel sont pourtant des obrtacles importants pour l'accès de l'utilisateur final aux données. Dans une situation réelle, le modèle des données est très large et contient plusieurs dizaines d'entités. Les bases sont alors constituées de nombreuses tables, reliées entre elles par divers liens dont le sens n'est pas toujours explicite. Souvent, les organisations ont choisi une norme pour définir des noms à chaque objet (table, champ,...) très « syntaxiques » sans sémantique claire. Le but était de faciliter le travail des développeurs et l'efficacité des procédures. L'utilisateur final n'est pas considéré ici car c'est à l'informaticien de lui proposer une abstraction de ces modèles à travers les outils dont il a besoin. La complexité des données, l'absence d'annuaire clair rend la base inutilisable aux non initiés sans l'intervention d'informaticiens et d'outils sur mesure.

L'informatique de production a donc été conçue pour privilégier les performances de tâches répétitives, prévues et planifiées tournées vers la prnduction de documents standards (factures, commandes...). L'intervention de l'utilisateur est guidée à travers des outils spécifiques proposés par une équipe de développeurs.

La dernière caractéristique de ces bases de données est qu'elles conservent l'état instantané du système. Dans la plupart des cas, l'évolution n'est pas conservée. On conserve simplement des versions instantanées pour la reprise en cas de panne et pour des raisons légales.

1.3.2 Le décisionnel

Dans un système d'information décisionnel, l'utilisateur final formulera des questions du type :

- Comment se comporte le produit X par rapport au produit Y ?
- Et par rapport à l'année dernière ?
- Quel type de client peut bien acheter mon produit Z ?

Ces exemples permettent de mettre en évidence les faits suivants :

- les questions doivent pouvoir être formulées dans le langage de l'utilisateur en fonction de son « métier », c'est-à-dire de son secteur d'activité (service marketing, service économique, service gestion des ressources humaines, ...) .
- la prévision des interrogations est difficile car elles sont du ressort de l'utilisateur. De plus, ses questions vont varier selon les réponses obtenues : si le produit X s'est vendu moins bien que l'année précédente, il va être utile d'en comprendre les raisons et donc de détailler les ventes du produit X (par région, par type de magasin, ...).
- des questions ouvertes (profil client du produit Z) vont nécessiter la mise en place de méthodes d'extraction d'informations.

Ce qui caractérise d'abord les besoins, c'est donc la possibilité de poser une grande variété de questions au système, certaines prévisibles et planifiées comme des tableaux de bord et d'autres imprévisibles. Si des outils d'édition automatiques pré-programmés peuvent être envisagés, il est nécessaire de permettre à l'utilisateur d'effectuer les requêtes qu'il souhaite, par lui-même, sans intervention de programmeurs. Deux contraintes apparaissent alors immédiatement : la simplicité du modèle des données, la performance malgré les grands volumes.

1.4 Architecture de systémes decisionnelles

On qualifie d'informatique décisionnelle (en anglais "Business intelligence", parfois appelé tout simplement "le décisionnel") l'exploitation des données de l'entreprise dans le but de faciliter la prise de décision par les décideurs, c'est-à-dire la compréhension du fonctionnement actuel et l'anticipation des actions pour un pilotage éclairé de l'entreprise.

Les outils décisionnels sont basés sur l'exploitation d'un système d'information décisionnel alimenté grâce à l'extraction de données diverses à partir des données de production, d'informations concernant l'entreprise ou son entourage et de données économiques.
Un outil appelé **ETL** (Extract, Transform and Load) est ainsi chargé d'extraire les données dans différentes source, de les nettoyer et de les charger dans une base de données géante (Entrepôt de données). Lors des premiers projets décisionnels, la phase de la collecte des données et de leur transformation était souvent sous-estimée. C'est peut-être là une des principales explications Des échecs de réalisations et des très nombreux dépassements de budget. Pourtant cette phase d'extraction est de transformation préalable représente (selon les spécialistes du milieu) à peu près les trois quart du projet de création d'un entrepôt de données.
Les outils ETL (Extract Transform Load) prennent en charge l'une des fonctions les plus essentielle du système globale décisionnel. Il s'agit en effet de gérer toutes les étapes préalables au chargement effectif des données dans l'etrepôt de données.
Extraire : Accéder à la majorité des systèmes de stockage de données (Bases de donées relatioelles, bases prioritaires, fichiers à plat...) et récupérer les données identifiées.
Transformer : Transformer, reformater et nettoyer les données afin d'éliminer les valeurs non conforme au modèle de destination et d'éviter les doublons et autres incohérences.
Charger : Insérer les données dans l'entrepôt de données. Où elles sont mise à disposition des outils d'analyse Data Mining, l'analyse multimensionnelle, etc.

La figure FIG 1.1 représente une architecture de système décisionnel très utilisée. Dans cette architecture, on dispose d'un entrepôt de données. L'entrepôt centralise les données issues de plusieurs sources (bases de production de l'entreprise, fichiers textes, documents web, etc). Ces données sont fusionnées dans l'entrepôt qui est généralement une grosse base de données. Ensuite, une fois l'entrepôt confectionné, des données sont extraites dans des serveurs d'analyse afin d'être analysées. Enfin des générateurs d'états sont utilisés afin de présenter l'étude aux utilisateurs finaux ou décideurs.

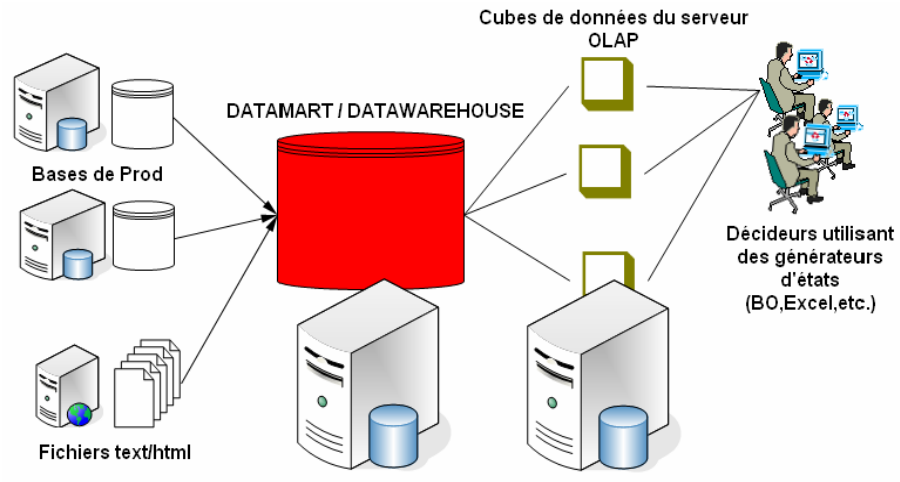

FIG. 1.1 – Processus de systéme décisionnelle

Les entrepôts de données permettent de produire des rapports qui répondent à la question « Que s'est-il passé ? », mais ils peuvent être également conçus pour répondre à la question analytique « Pourquoi est-ce que cela s'est passé ? » et à la question pronostique « Que va-t-il se passer ? ». Dans un contexte opérationnel, ils répondent également à la question « Que se passe-t-il en ce moment ? », voire dans le cas d'une solution d'entrepôt de données actif « Que devrait-il se passer ? ».

Le reporting est vraisemblablement l'application l' plus utilisée encore aujourd'hui de l'informatique décisionnelle, il permet aux gestionnaires de :

1. Sélectionner des données relatives à telle période, telle production, tel secteur de clientèle, etc.,
2. Trier, regrouper ou répartir ces données selon les critères de leur choix,
3. Réaliser divers calculs (totaux, moyennes, écarts, comparatif d'une période à l'autre, ...),
4. Présenter les résultats d'une manière synthétique ou détaillée, le plus souvent graphique selon leurs besoins ou les attentes des dirigeants de l'entreprise.

Les programmes utilisés pour le reporting permettent bien sûr de reproduire de période en

période les mêmes sélections et les mêmes traitements et de faire varier certains critères pour affiner l'analyse. Mais le reporting n'est pas à proprement parler une application d'aide à la décision. L'avenir appartient plutôt aux instruments de type tableau de bord équipés de fonctions d'analyses multidimensionnelles de type **OLAP**.
Les datamart et/ou les entrepôts de données peuvent ainsi permettre via l'OLAP l'analyse très approfondie de l'activité de l'entreprise, grâce à des statistiques recoupant des informations relatives à des activités apparemment très différentes ou très éloignées les unes des autres, mais dont l'étude fait souvent apparaître des dysfonctionnements, des corrélations ou des possibilités d'améliorations très sensibles.

1.5 Techonologie OLAP :

Le but de l'**OLAP** (*On-Line Analytical Processing*) est de permettre une analyse multidimensionnelle sur des bases de données volumineuses afin de mettre en évidence une analyse particulière des données (il est l'objet d'un questionnement particulier). Elle désigne une catégorie d'applications et de technologies permettant de collecter, stocker, traiter et restituer des données multidimensionnelles, à des fins d'analyse.
Grâce à l'OLAP, les utilisateurs peuvent créer des représentations multidimensionnelles (appelées hypercubes ou "cubes OLAP"(selon les critères qu'ils définissent afin de simuler des situations.
Un hypercube OLAP (ou cube OLAP) est une représentation abstraite d'informations multidimensionnelles exclusivement numérique utilisé par l'approche OLAP.
Cette structure est prévue à des fins d'analyses interactives par une ou plusieurs personnes (souvent ni informaticiens ni statisticiens) du métier que ces données sont censées représenter. Les cubes OLAP ont les caractéristiques suivantes :

1. Permet de simuler le comportement d'un SGBD multidimensionlel.
2. Obtenir des informations déjà agrégées selon les besoins de l'utilisateur.
3. Simplicité et rapidité d'accès.
4. Capacité à manipuler les données agrégées selon différentes dimensions.

5. Plus facile et moins cher à mettre en place.

Ces astuces notées dans ce pargraphe deviendront plus claires dans le chapitre suivant où nous allons détailler la représentation multi-dimensionnelle.

1.5.1 Manipulation de données multidimensionnelles

L'hypercube OLAP donne accès à des fonctions d'extraction de l'information (pour visualisation, analyse ou traitement), et à des fonctions de requête en langage MDX (comparable à SQL pour une base de données relationnelles). Citons les operation de la manipulation des données multidimensionnelles :

- **Rotate** : sélection du couple de dimensions à cibler.
- **Slicing** : extraction d'une tranche d'information.
- **Scoping** : extraction d'un bloc de données (opération plus générale que le slicing).
- **Drill-up** : synthèse des informations en fonction d'une dimension.
- **Drill-down** : c'est l'équivalent d'un « zoom », opération inverse du drill-up.
- **Drill-through** : lorsqu'on ne dispose que de données agrégées (indicateurs totalisés), le drill through permet d'accéder au détail élémentaire des informations.

1.5.2 Le MDX

Comme les bases de données relationnelles, la représentétion muliti dimensionnelle peut être implimentée sur plusieurs logiciels crées pour ce domaine. Le **MDX** est parmi les logiciels associés à la manipulation des données multidimensionnelles.
Le **MDX** est le langage permettant de définir, d'utiliser et de récupérer les données à partir d'objets multidimensionnels et permet d'effectuer les opérations décrites précédemment, il est equivalent à SQL pour le monde OLAP. Ce logiciel est une fondation originale de la maison **Microsoft** .

1.5.3 OLAP vs OLTP

Le tableau suivant donne une comparaison entre l'OLAP et l'OLTP, et résume la différence entre eux :

OLTP	**OLAP**
Orienté transaction	Orienté analyse
Orienté application	Orienté sujet
Données courantes	Données historisées
Données détaillées	Données agrégées
Données évolutives	Données statiques
Utilisateurs nombreux, administrateurs/opérationnels	Utilisateurs peu nombreux, manager
Temps d'exécution : court	Temps d'exécution : long

1.6 Les entrepôts de données

Le concept d'entrepôt de données (Data Warehouse) a été formalisé pour la première fois en **1990** par **Bill Inmon**. Il s'agissait de constituer une base de données orientée sujet, intégrée et contenant des informations historisées, non volatiles et exclusivement destinées aux processus d'aide à la décision.

En effet, la simple logique de production (produire pour répondre à une demande) ne suffit plus pour pérenniser l'activité d'une entreprise. Elle est un système ouvert sur son environnement au coeur des systèmes d'informations confrontée à des phénomènes économiques et sociaux lourd de conséquences.

Pour faire face aux nouveaux enjeux, l'entreprise doit collecter, traiter, analyser les informations de son environnement pour anticiper. Mais cette information produite par l'entreprise est surabondante, non organisée et éparpillée dans de multiples systèmes opérationnels hétérogènes et peut provenir de toutes les places de marchés (mondialisation des échanges).

Il devient fondamental de rassembler et d'homogénéiser les données afin de permettre l'analyse des indicateurs pertinents pour faciliter la prise de décisions. L'objet de l'entrepôt de données est de définir et d'intégrer une architecture qui serve de fondation aux applications décisionnelles.

1.7 Définition

Bill Inmon, père fondateur du Data Warehouse en donne une définition :
« L'entrepôt de données est une collection de données orientées sujet (thématiques), intégrées, non volatiles et historisées, organisées pour le support d'un processus d'aide à la décision » expliquons ces quatre mots clés de cette définition :

1. **Données thématiques :** La vocation d'entrepôt de données est de prendre des décisions autour des activités majeures de l'entreprise. Les données sont ainsi structurées autour de thèmes ce qui facilite l'analyse transversale. Pour éviter le doublonnage des données, on regroupe les différents sujets dans une structure commune. Ainsi, si le sujet client contient des informations dans les sujets marketing, ventes, analyse financière, on regroupera ces trois sujets au sein du thème client. Dès lors, chaque donnée n'est présente qu'à un endroit et l'entrepôt de données joue bien un rôle de point focal.

2. **Données intégrées :** Les données sont mises en forme selon un standard afin d'obtenir la transversalité recherchée. Cela nécessite une forte normalisation, une bonne gestion des référentiels et de la cohérence, une parfaite maîtrise de la sémantique et des règles de gestion des données manipulées.
   Lors de l'alimentation des données, ces dernières sont hétérogènes et proviennent de systèmes opérationnels différents. Il faut doter ces données d'une codification unique et pertinente afin qu'elles puissent aisément s'intégrer dans le Data Warehouse. Il faudra donc faire appel à des conventions de nomage, des structures de codage, qualifier les mesures et réaliser l'intégration de la sémantique

3. **Données non volatiles :** Les données intégrées dans l'entrepôt ne peuvent subir aucune altération. Cela se justifie pour assurer la fiabilité des résultats des requêtes. Ainsi, une même requête lancée à plusieurs mois d'intervalle donnera toujours les mêmes résultats. Cela permet au Data Warehouse d'acquérir au cours du temps un historique détaillé de l'activité de l'entreprise. Ceci s'oppose fondamentalement à la logique des systèmes de production qui remettent à jour les données qui sont de nature volatiles.

4. **Données historisées :** L'ensemble des données qui sont intégrées dans l'entrepôt contient un ensemble de caractéristiques qui sont datées. L'historisation est nécessaire pour suivre dans le temps l'évolution des différentes valeurs des indicateurs à analyser. Ainsi, un référentiel temps doit être associé aux données afin de permettre l'identification dans la durée de valeurs précises. Si tel n'était pas le cas, l'analyse ne serait pas possible, le suivi de l'évolution non plus. Cela rejoint la notion de non volatilité expliquée précédemment.

1.8 Le modéle multidimentionnel

Le modèle conceptuel doit être simplifié au maximum pour permettre au plus grand nombre d'utilisateurs d'appréhender l'organisation des données et de comprendre ce que l'entrepôt de données mémorise. On parle de modèle multidimensionnel. Ce dernier est utilisé dans les applications dont l'objectif est d'analyser les données plutôt que de procéder à des transactions on-line.
La technologie des bases de données multidimensionnelles est un facteur clé dans l'analyse de larges quantités de données dans l'informatique décisionnelle. En effet, contrairement aux technologies précédentes, les données sont vues comme des cubes particulièrement adaptés à l'analyse de données dans le modèle multidimensionnel.

1.8.1 Les cubes

Dans le modèle multidimensionnel, les données seront toujours des faits à analyser suivant plusieurs dimensions. Par exemple, dans le cas de ventes de produits à des clients dans le temps (trois dimensions, un vrai cube), les faits sont les ventes, les dimensions sont les clients, les produits et le temps. Cela revient à dire que pour chaque combinaison des trois dimensions (clients, produis, temps), ont peut accéder à la mesure numérique associée au fait vente (cellule non vide). Les interrogations s'interprètent souvent comme l'extraction d'un plan, d'une droite de ce cube (par exemple, lister les ventes du produit A ou lister les ventes du produit A sur une période de temps D), ou l'agrégation de données le long d'un plan ou d'une droite (par exemple, obtenir le total des ventes du produit A

revient à sommer les éléments du plan indiqué en figure FIG 1.2).

Un entrepôt de données est composé d'un ensemble de cubes reliés. Théoriquement, un

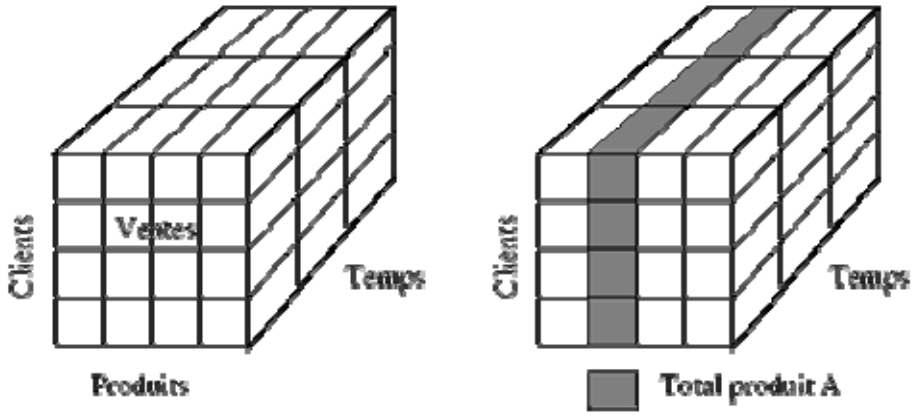

FIG. 1.2 – Cube de données

cube peut contenir un nombre infini de dimensions. Mais dans le monde réel, la plupart des cubes contiennent quatre à douze dimensions. Des problèmes de performance sont observés au-delà de cet intervalle.

Un avantage évident de ce modèle est la simplicité qu'il représente dès que les mots ventes, clients, produits et temps sont précisés. Il reprend les termes et la vision de l'entreprise de tout utilisateur final concerné par les processus de décision.

1.8.2 Les faits

Un fait représente un sujet d'analyse. Il est constitué de plusieurs mesures relatives au sujet traité. Ces mesures sont numériques et généralement valorisées de façon continue.

Dans la plupart des modèles multidimensionnels, les faits sont implicitement représentés par la combinaison des valeurs des dimensions. Un fait n'existe que si une combinaison particulière des dimensions découle vers une cellule non vide du cube le représentant.

1.8.3 Les dimensions

Les dimensions sont un concept essentiel dans les bases de données multidimensionnelles. Elles sont les critères suivant lesquels on souhaite évaluer, quantifier et qualifier le fait. Elles peuvent être utilisées pour la sélection des données selon le niveau de précision désiré. Aussi, elles peuvent être affinées, décomposées en hiérarchies, afin de permettre à l'utilisateur d'examiner ses indicateurs à différents niveaux de détail, de "descendre" dans les données, en allant du niveau global au niveau le plus fin. Par exemple, une date pourra être décomposée en <année, mois, semaine, jour>. On aura alors une vision pyramidale des données, la base de la pyramide représentant le niveau le plus détaillé et le haut le niveau le plus global.

L'utilisateur peut avoir besoin de personnaliser le modèle défini par l'administrateur en

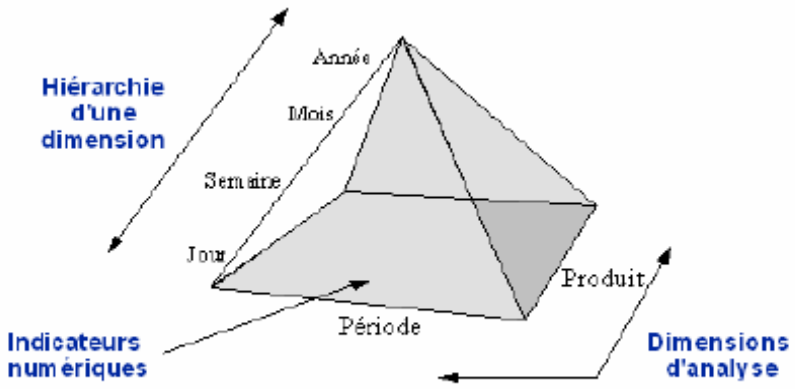

FIG. 1.3 – Représentation pyramidale

incorporant par exemple ses propres attributs dans les dimensions ou en modifiant certaine hiérarchies.

1.9 Modélisation

Partant du principe que les données sont des faits à analyser selon plusieurs dimensions, il est possible de réaliser une structure de données simple qui correspond à ce besoin de

modélisation multidimensionnelle. Cette structure est constituée du fait central et des dimensions.

Au niveau logique, cela peut se traduire par trois modèles différents : en étoile, en flacon de neige ou en constellation. Nous nous basons dans notre étude sur le modèle etoilé car c'est le plus simple.

1.9.1 Modèle en étoile

Le centre est la table des faits, et les branches en sont les dimensions. Pour une dimension, il existe plusieurs faits (FIG 1.4). La structure est dissymétrique. en effet, la table des faits est énorme et les tables de dimensions sont petites.

La table de faits contient une clé composée des clés des tables de dimensions.

Les tables de dimensions contiennent quant à elle une clé primaire unique correspondant exactement à l'un des composants de la clé de la table des faits. Les faits sont généralement numériques alors que les dimensions sont qualitatives, elles contiennent des informations textuelles pour décrire les faits.

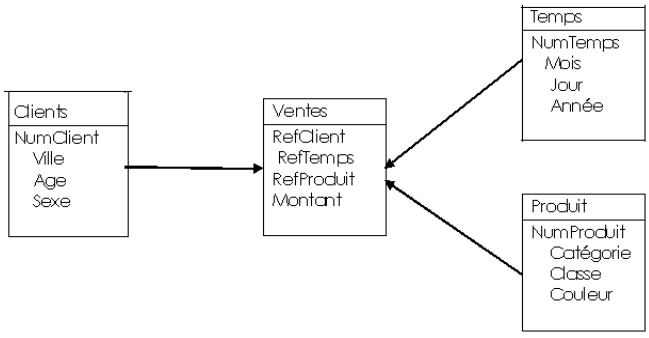

FIG. 1.4 – Modéle en étoile

1.9.2 Modèle en flocon

Le principe est le même que pour le modèle en étoile, mais en plus, les dimensions sont décomposées. Le but est d'économiser ainsi de la place. Cela permet également d'instaurer une hiérarchie au sein des dimensions, mais engendre par contre une complexification du modèle.

1.9.3 Modèle en constellation

Il est encore basé sur le modèle en étoile, mais on rassemble plusieurs tables des faits qui utilisent les mêmes dimensions.

1.10 L'alimentation des entrepôts de données

L'alimentation est la procédure qui permet de transférer des données du système opérationnel vers l'entrepôt de données en les adaptant. La conception de cette opération est une tâche assez complexe. Elle doit être faite en collaboration avec les administrateurs des bases de production qui connaissent les données disponibles et vérifient la faisabilité des demandes.

Dans cette étape, il est nécessaire de déterminer :

- Les données chargées dans l'etrepôt, on appelle ça la **matérialisation de vues**.
- Les transformations et vérifications nécessaires, la périodicité et le moment auxquels les transferts des donées auront lieu, on appelle ça la **maintenance de l'entrepôt de données**.

Traditionnellement, ce sont en grande partie des faits qui seront ajoutés. Les dimensions évoluent selon leur nature soit peu soit assez rapidement. Nous pouvons par exemple imaginer que la gamme des produits n'est pas très souvent modifiée. Par contre, de nouveaux clients seront certainement ajoutés mais bien moins que des dates. Le chargement implique aussi des problèmes physiques. L'alimentation est ponctuée d'une procédure d'assurance qualité qui veillera à ce que tout se soit déroulé correctement avant la publication du nouveau jeu de données.

1.10.1 Les vues matérialisées

Pour les énormes entrepôts de données, il est nécessaire d'avoir recours à des méthodes efficaces d'accès aux données pour répondre aux requêtes de manière optimale.
Une alternative possible est d'utiliser des structures redondantes. En effet, parmi les techniques issues des implémentations relationnelles des entrepôts de données, la matérialisation de vues est très efficaces pour améliorer le temps de réponse des requêtes décisionnelles.
Ainsi, un des aspects les plus importants dans la conception physique de l'entrepôt est justement de sélectionner un ensemble approprié de vues à matérialiser, qui permet de minimiser le temps de réponse des requêtes, compte-tenu d'un **espace de stockage** limité car l'entrepôt a un volume de stockage donné, et les vues ne doivent pas dépasser ce volume.
Nous distinguons deux types de vues :

1. **Vues virtuelles :** les données de la vue restent physiquement stockéees au niveau des sources et la vue est calculée au moment de l'interrogation.
2. **Vues matérialisées :** la vue est calculée avant l'interrogation et les données sont physiquement stockées dans l'entrepôt.

Le choix de ces vues à matérialiser présente un problème ; une vue matérialisée permet de diminuer le temps d'interrogation mais elle nécessite un coût de rafraîchissement et de stockage.
Dans notre étude la contrainte de maintenance est prise en compte, alors que la contrainte de stockage ne pose pas vraiment un grand problème, tel que il existe plusieurs méthodes pour résoudre le problème de l'alimentation de l'entrepôt de données sous cette contrainte, et elles donnent des résultats excéllentes et performantes.

1.10.2 La maintenance de vues

Un entrepôt de données contient un ensemble des vues matérialisées dérivées à partir de tables qui ne résident pas dans l'entrepôt de données.
Les tables de base changent et évoluent à cause des mises à jour.

Cependant, si ces changements ne sont pas reportés dans les vues matérialisées, leurs contenus deviendront obsolétes et leurs objets ne représenteront plus la réalité.
Par conséquent, un objet d'une vue peut continuer à exister alors que les objets à partir lesquels il a été dérivé ont été supprimés ou modifiés.
A fin de résoudre ce probléme d'inconsistance des données, une procédure de maintenance des vues doit être mise en place.

Les stratégies de la maintenance de vues : Trois stratégies fondamentales ont été proposées, pour la maintenance de vues :

- **Mise à jour périodique :** Dans cette stratégie, les vues sont mises à jour périodiquement . Dans ce cas, ces vues peuvent être considérées comme des photographies (snapshots).
- **Mise à jour immédiate :** C'est la plus coûteuse, dans cette stratégies, les vues sont mises à jour immédiatement aprés chaque transaction.
- **Mise à jour différée :** Dans cette derniére stratégie, les modifications sont propagées d'une maniére différée. Dans ce cas, une vue est mise à jour uniquement au moment où elle est utilisée par une requête d'un utilisateur.

En pratique, la mise à jour périodique est l plus appliquée, telle qu'on s'occupe de maintenir les vues aprés chaque période donnée . Donc, la premiére question qui peut se poser est : quelle est la valeur de cette période ?
La réponse sur cette question repose sur la résolution d'un probléme intéressant, ce probléme est la détermination de ce qu'on appélle la police de maintenance .
Plusieurs recherches ont été entamées dans la détermination de la police de maintenance, même plusieurs algorithmes ont été proposés par les chercheurs (Informaticiens, Mathématiciens d'applications), ce qui prouve que ce probléme en réalité est un probléme dure.

1.11 Conclusion

Dans ce chapitre nous avons vu la différence entre l'informatique de production et l'informatique de décision.
Nous avons exposé l'architecture des systémes decisionnels souvent utilisés en littérature.

Enfin nous avons parlé de la technologie **OLAP** qui permet l'analyse des systémes décisionnels.

On s'est intéréssé aussi à la définition des entrepôts de données, ainsi que leurs conceptions.

Nous avons présenté le concept d'alimentation des entrepôts de données, ainsi que les tâches nécessaires pour l'efectuer (matérialisation de vues, maintenance de vues, etc...).

Dans le prochain chapitre, nous parlerons de la matérialisation de vues sous contraintes de ressources. On se basera sur la contrainte de maintenance.

Chapitre 2

La matérialisation sous contrainte de la maintenance

2.1 Introduction

Dans la plupart des applications de matérialisation de vues dans les entrepôts de données, le but est de minimiser le temps de réponse des requêtes afin de performer ces entrepôts et les rendre plus efficaces, mais c'est pas toujours le cas à cause de certaines contraintes qui privent de stocker les vues de la solution optimale dans l'entrepôt de données.

L'entrepôt de données est caractérisé par son volume qui détermine sa capacité de stockage, donc c'est pas toujours possible de matérialiser toutes les vues surtout dans le cas où le nombre de requêtes est trés elevé .

Il est carcterisé aussi par son temps de maintenance, ce temps est peut être défini comme la période qu'il ne faut pas dépasser pendant la maintenance de vues.

Le problème de matérialisation de vues sous la contrainte de capacité de stockage semble plus facile que le problème sous la contrainte de temps de maintenance à cause d'une propriété qu'on va expliquer au cours de ce chapitre.

2.2 L'approche de la sélection de vues materialisées :

Soit le schémas suivant :

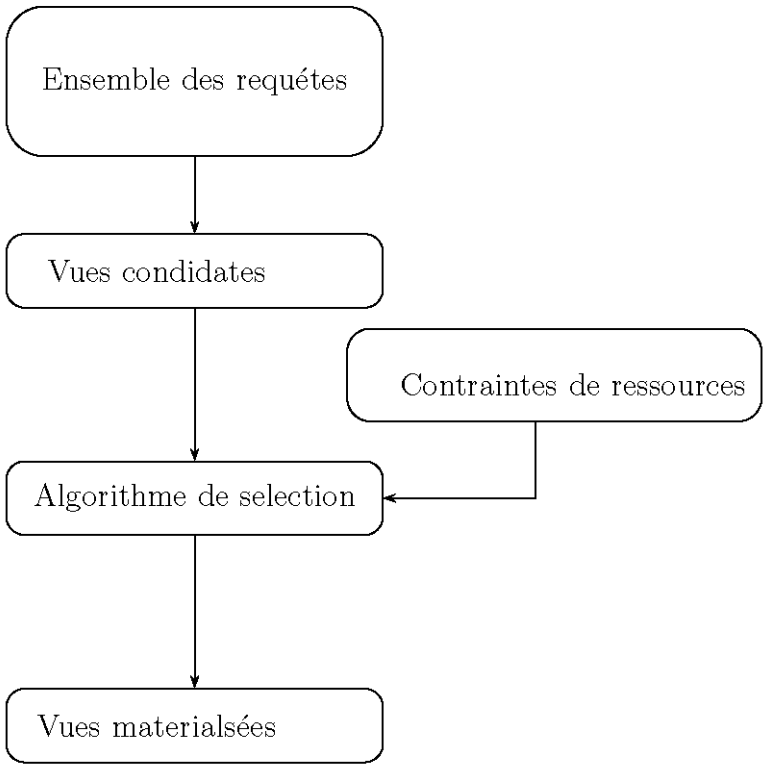

FIG. 2.1 – L'approche de selection

Dans l'environnement d'un entrepôt de données, il est généralement possible d'isoler un ensemble de requêtes à privilégier. L'ensemble de vues matérialisées doit être déterminé en fonction de cet ensemble de requêtes.
Ce probléme a été largement étudié afin de sélectionner un ensemble de vues optimal résultant d'une énumération de toutes les vues à matérialiser (ou à précalculer).
La recherche d'un algorithme efficace pour la matérialisation des vues est un sujet de recherches de plusieurs informaticiens, mais avec l'apparaition des nouvelles techniques connues dans le domaine de la recherche operationnelle, ce problème d'optimisation est devenu parmi les applications les plus intéressantes, et la preuve est que plusieurs mathématiciens ont proposés des algorithmes basés sur des notions de la recherche opérationnelle

pour performer les résultats existants ou créer de nouvelles techniques d'optimisation.
Le probléme peut être détaillé de la façon suivante :
Étant donné une contrainte de ressource S (capacité de stockage, coût de maintenance), le problème consiste à sélectionner un ensemble de vues $\{V_1, V_2, ..., V_k\}$ minimisant une fonction objectif (coût total de traietment des requêtes, coût de maintenance de vues sélectionnées) et satisfaisant la contrainte.

Proposition :

Le problème de matérialisation de vues sous contrainte de maintenance est $NPComplet$.

Preuve :

Premiérement rappelons le probléme du sac à dos.
On dispose de n objets ayant chacun un poid a_j et une valeur $c_j (j = 1, 2, ..., n)$. Il faut effectuer une sélection (déterminer un sous ensemble de ces n objets) dont le poids total soit inférieur ou égal à un nombre donné et dont la valeur, somme des valeurs des objets sélectionnés, soit maximum.
Le problème peut se formuler de la maniére suivante :
Il s'agit de déterminer $J \subset \{1, 2, ..., n\}$ tel que :
$\sum_{j \in J} c_j$ soit maximum,
Sous la contrainte :
$\sum_{j \in J} a_j \leq b$
Si on pose $S = \{v_1, v_2, ..., v_n\}$ l'ensemble de vues à matérialiser, et $s(v_i)$ la différence entre le coût de maintenance aprés et avant l'ajout de la vue v_i. $Profit(v_i)$ le coût de traitement aprés l'ajout de la vue v_i.
Á partir du problème du sac à dos, Le nombre de vues est le nombre d'objets n, et pour chaque objet on associe une vue. Pour une vue v_j avec $(j = 1, 2, ..., n)$, on pose la taille a_j et la valeur c_j sont respectivement $s(v_i)$ et $Profit(v_i)$.
Il résulte d'une maniére directe que le probléme de la matérialisation de vues est une

réduction polynômiale directe à partir de problème du sac à dos.
La sélection de toute les vues nous garantie une solution optimale mais ce cas n'est pas toujours réalisable à cause de l'espace de stockage de ces vues dans l'entrepôt qui peut être insuffisant pour garder toutes les vues. Une deuxième contrainte aussi trés intéressante est la contrainte de maintenance de vues qui est le sujet de notre étude.

2.3 Représentation des relations entre les vues

Les utilisateurs d'entrepôts de données travaillent dans des environnements graphiques, et les données ont une représentation graphique. Parmi ces modéles de représentation on trouve le modèle dimensionnelle décrit ci-dessous.
Pour donner plus de précisions, reprenons l'exemple précèdent :
La structure de treillis (FIG 2.2) est introduite en vue de représenter les dépendances

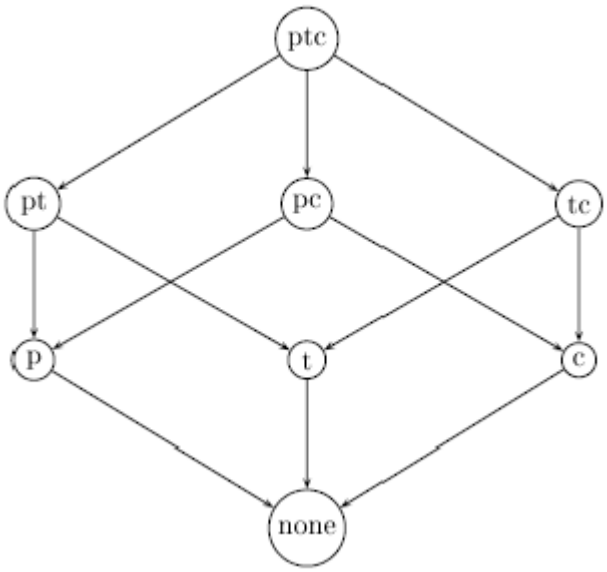

FIG. 2.2 – La dependance entre les vues

entre les vues à matérialiser. En algèbre, le treillis est un ensemble ordonné où chaque couple d'éléments admet un infimum et un supremum.
Un treillis de vues est un graphe qui représente la relation hiérarchique entre toutes les vues. Les noeuds représentent les vues (ou requêtes) et les arcs représentent les relations

de dépendance entre les vues.

Pour deux sommets x, y du graphe, il existe un chemin de x vers y, si et seulement si la vue x répond à la vue y.

Les cellules du treillis sont organisées dans différentes séries, par exemple dans la figure figure 2.2, (p, ALL, t) contient le prix total de ventes d'un produit p acheté dans un temps donné t en tenant compte de tous les clients. La valeur de la cellule (p, ALL,t) peut être calculée à partir de la somme des valeurs des cellules (p,c_1, t), (p, c_2, t), ..., (p,c_n,t), où n est le nombre de clients. Chaqu'une de ces cellules correspond à une requête SQL, ce qui veut dire que le noeud *pct* répond au noeud *pc* ou d'une maniére équivalente le noeud *pc* est dependant du noeud *pct*

On dit qu'une vue x est un parent de la vue y et y est un enfant de x si la vue x répond à la vue y et il n'existe aucune vue z tel que x répond à z et z répond à y. Si z existe dans le graphe représentatif entre x et y, x est un ascendant de y et y est un descendant de x.

2.4 Modélisation mathématique du problème

Reprenons l'exemple de la section précédente. Les noeuds du treillis représentent les requêtes OLAP et les arcs représentent les dépendances entre ces requêtes. On note ce graphe par $(L, \leq)$, avec :

L : représente l'ensemble des vues,

$\leq$: représente la relation de dépendance entre les requêtes.

Etant donné deux requêtes Q_i et Q_j, on dit que Q_i dépend de Q_j que l'on note $Q_i \leq Q_j$ si on peut répondre à la requête Q_i en utilisant la requête Q_j.

Le graphe de cette représentation est un graphe acyclique noté $G = (V, U)$, tel que V représente l'ensemble des noeuds de G et U représente l'ensemble des arcs de G.

Un arc $v_j v_i$ existe dans U si et seulement si $v_i \leq v_j$ et $\nexists v_k$ tel que $v_i \leq v_k$ et $v_k \leq v_j$ pour $v_i \neq v_k \neq v_j$.

Ce graphe est un graphe pondéré à la fois sur les sommets et sur les arcs. Un sommet donné v est caractérisé par trois poids :

r_v : Le coût initial de la requête v,

f_v : La fréquence de la requête v,

g_v : La fréquence de mise à jour de la requête v .

Pour les arcs nous distinguons deux poids :

q_{uv} : le coût de traitement de la requête u en utilisant la requête v,

m_{uv} :le coût de maintenance de la requête u en utilisant la requête v.

Notons que la source est prise en considération dans le cas où les deux vues sont indépendantes, cette source est appellé aussi la vue vertuelle.

Soit $ct(u, v)$ le coût de traitement de la requête u en utilisant la requête v, $ct(u, v)$ est la somme des coûts de traitements associés sur les arcs de la plus courte chaine de v vers u, plus la valeur de r_v. Si v ne répond pas à u on répond à u à partir de la source. Soit $cm(u, v)$ le coût de maintenance de la requête u en utulisant la requête v, $cm(u, v)$ est la somme des coûts de maintenances associés sur les arcs de la plus courte chaine de v vers u. Si v ne répond pas à u on maintient u à partir de la source.

Soit $q(v, M)$ le coût de traitement minimum de la vue v en présence de l'ensemble de vues materialisées M, et $m(v, M)$ le coût de maintenance minimum de la vue v en présence d'ensemble de vues materialisées M.

Donc le problème peut être formulé de la façon suivante : Choisir l'ensemble de vues M qui minimise le coût de traitement total des requêtes noté en présence d'ensemble de vues à materialiser M noté $\tau(G, M)$ avec

$$\tau(G, M) = \sum_{v \in V} f_v * q(v, M).$$

Sous la contrainte :

$$U(M) = \sum_{v \in M} g_v * m(v, M) \leq S.$$

S est le seuil maximum.

2.5 Notion du bénéfice :

Soit $G = (V, E)$ le graphe représentatif des relations entre les vues avec V est l'ensemble des vues et E l'ensemble d'arcs de la dépendance entre les vues .

2.5.1 La fonction du bénéfice :

Soit $\tau(G, M)$ qui représente le coût de traitement total pour répondre à toutes les requêtes en utilisant les vues matérialisées dans l'ensemble M.
Soit $U(M)$ le coût de maintenance de la matérialisation de l'ensemble de vues dans M.
Dans une étape donnée, l'ensemble de vues materialisées est M.
Si on veut ajouter une autre vue notée v le temps de traitement de ce nouvel ensemble va changer à $\tau(G, M \cup v)$ et le coût de maintenance devient $U(M \cup v)$
Nous représentons ici une nouvelle fonction $g(v)$, qui est en fonction de la nouvelle vue materialisée v telle que :

$$g(v) = \frac{\tau(G, M) - \tau(G, M \cup v)}{U(M \cup v) - U(M)}.$$

2.5.2 La difficulté de problème sous contrainte de maintenance :

Dans l'expression de $g(v)$ la valeur de $B(G, M) = \tau(G, M) - \tau(G, M \cup v)$ est toujours strictement positive, le coût de traitement va diminuer avec $B(G, M)$ après la matérialisation d'une nouvelle vue v.
Mais c'est pas la même chose pour la fonction $\delta T_v = U(M \cup v) - U(M)$, car δT_v peut augmenter et peut diminuer aussi. Donc après la matérialisation d'une vue v, le gain $g(v)$ peut être positif ou négatif.
Le gain de matérialisation de vue par unité de coût de maintenance peut changer de signes soit positif ou négatif dans chaque étape de la matérialisation de vues, cette propriéte qui n'existe pas dans le problème de la materialisation de vues dans un entrepôt de données sous contrainte de capacité de stockage, tel que le gain de vues par unité de stockage est toujours positif.

Sans plus de details il est évident de comprendre que le gain de la matérialisation d'une vue v par unité d'espace de stockage est donné par :

$$g(v) = \frac{\tau(G, M) - \tau(G, M \cup v)}{ST(M \cup v) - ST(M)}.$$

La valeur de $ST(M \cup v) - ST(M)$ restera toujours positive, en effet, le volume occupé augmente avec l'ajout des vues supplimentaires.
Cette différence entre les problèmes de matérialisation sous la contrainte de volume de stockage et le temps de maintenance est trés sensible.
Cette propriété est nommée la monotonicité. On dit que le gain de la matérialisation de nouvelle vue est monotone par unité de stockage et non monotone par unité de coût de maintenance.
Cette non-monotonicité dans le cas de coût de maintenance rend la résolution de ce problème plus dure que dans le cas de contrainte de volume de stockage, et les algorithmes de résolution du problème sous la contarainte de volume donnent des solutions plus proches aux solutions optimales que dans le cas de contrainte de maintenance.

2.6 Algorithmes existants :

Dans cette section nous allons présenter quelques algorithmes connus dans le domaine de la matérialisation de vues sous la contrainte de maintenance. Il existe un certain nombre de tentatives pour résoudre ce problème, parmi elles on cite l'algorithme A^*, l'algorithme de deux phases et l'algorithme intégré.

2.6.1 L'algorithme A-star :

Cet algorithme a été réalisé par **Gupta** et **Mumick** en 1999 [2].
Commençons par citer cet algorithme :

Algorithm 1 A star

Entrées: Un graphe $G = (V, E)$ et contrainte de maintenance S ;

Sorties: Ensemble de vues materialisées ;

-Créer un arbre T_G ayant juste la racine A. L'étiquette associée à A est $< \emptyset, \emptyset >$;

-Créer une liste de priorité $L =< A >$;

Répéter

-Retirer x de L , telque $g(x) + h(x)$ est de valeur minimale dans L ;

-Soit l'etiquette de $x =< N_x, M_x >$, où $N_x = \{v_1, v_2, \ldots, v_d\}$ pour chaque $d \leq n$;

Si $d = n$ **Alors**

Recupérer M_x ;

FinSi

-Ajouter un successeur du x, $l(x)$, avec l'étiquette $< N_x \cup v_{d+1}, M_x >$ à la liste L ;

Si $U(M_x) < S$ **Alors**

-Ajouter à L un successeur de x, $r(x)$, avec une étiquette $< N_x \cup v_{d+1}, M_x >$;

FinSi

Jusqu'à $(L = \emptyset)$

Recupérer $\emptyset$

A-star est une heuristique qui utilise le tri topologique inverse afin de trouver un ensemble de vues matérialisées. Elle définit un arbre binaire T_G dont les feuilles sont les solutions candidates de ce problème. Â chaque étape de la recherche, A-star évalue le bénéfice des branches restantes vers le bas, et choisit la branche la plus bénéfique.

Chaque sommet dans cette arbre binaire de recherche a une étiquette $< N_x, M_x >$ avec $(M_x \subseteq N_x)$, où M_x est un ensemble des vues qui ont été choisis pour se matérialiser et qui est considéré pour répondre sur l'ensemble des requêtes N_x. L'espace de recherche est de $2^{|V(G)|}$, où $V(G)$ est l'ensemble des sommets du graphe G.

Nous estimons le bénéfice des branches descendantes en faisant la somme de deux fonctions $g(x)$ et $h(x)$. $g(x)$ correspond au coût de traitement total des requêtes N_x en utilisant les vues sélectionnés dans M_x. $h(x)$ est une limite inférieure estimé sur $h^*(x)$ qui est défini comme le coût de traitement restant d'une solution optimale correspondante à des

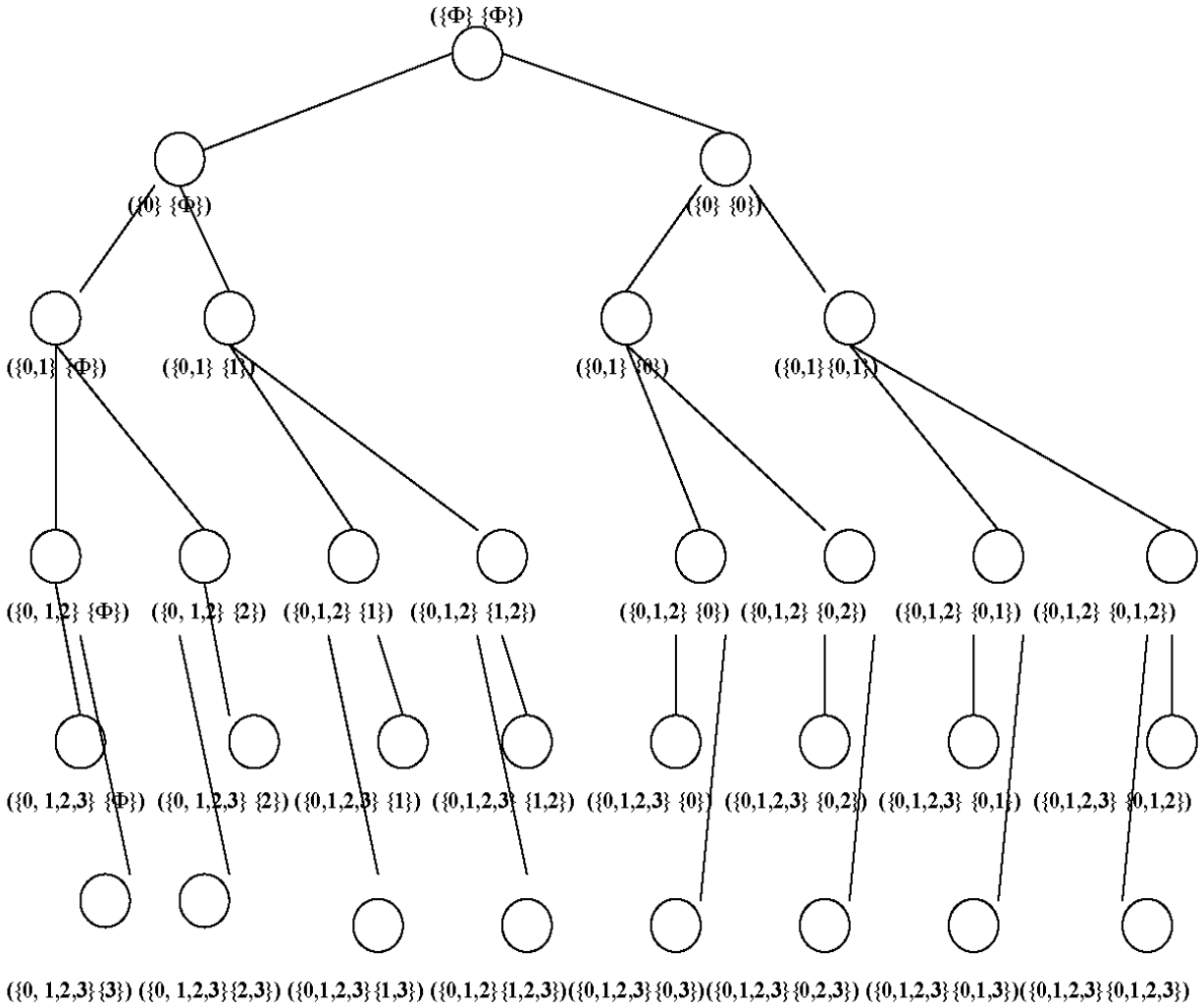

FIG. 2.3 – Exemple d'arbre de recherche de vues

descendants de x en T_G.

Cette heuristique donne des solutions exactes pour un ensemble de vues qui ne dépasse pas 32 vues c'est à dire pour une dimension maximale égale a 5. car la difficulté dans cet algorithme réside dans la détermination de la borne inférieur $h(x)$.

2.6.2 Algorithme des deux phases :

Cet algorithme a été réalisé par **Liang** et **Wang** et **Orlowska** en 2001 [3].

Algorithm 2 Algorithme des deux phases

Entrées: Un graphe $G = (V, E)$ qui représente la dépendance entre les vues, seuil de maintenance S ;

Sorties: Ensemble de vues à materialiser ;

-Trouver un ensemble de vues à matérialiser, M_1 , qui minimise le coût total de traitement de requêtes en utilisant un algorithme de résolution d'un problème de couplage de poids minimum et de cardinalité maximum ;

Si $U(M_1) \leq S$ **Alors**

-Récupérer M_1 et aller à 9 ;

Sinon

-Trouver un sous-ensemble de M_1, noté M_2, qui minimise le total des coûts du traitements de requêtes et répond à V en utilisant une sélection de vues par ordre décroissant de leur dimensions ;

-Récupérer M_2 ;

FinSi

L'algorithme des deux phases est illustré dans l'algorithme 2. L'idée de base consiste à choisir un sous-ensemble de vues à matérialiser, M_1, qui minimise le coût de traitement total sans tenir compte de la cotrainte de maintenance. Si le coût de maintenance total $U(M_1)$, des vues dans M_1, est inférieur ou égal au seuil maximum de maintenance S, alors, toutes les vues de M_1 seront matérialisées. Sinon, nous devons encore trouver un sous-ensemble de M_1, noté M_2, tel que :

1. On Peut répondre à toutes les vues de $M_1 - M_2$ en utilisant M_2
2. $U(M_2) \leq S$ avec coût de traitement minimum.

La premiére condition garantit qu'on peut répondre à toutes les requêtes, en utilisant les vues de M_2.

La deuxième condition garantit la contrainte de maintenance et le temps de réponse minimum.

Considerons un ensemble de quatre vues représenté dans le graphe suivant :

Dans la premiére phase, cet algorithme ne prend pas en considération la contarainte

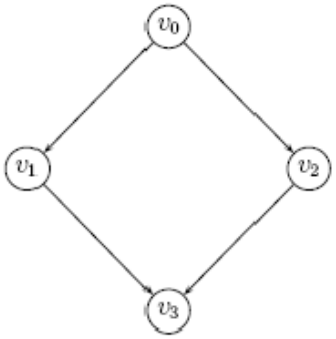

FIG. 2.4 – Exemple de dependance entre 4 vues

de maintenance, Il réduit ce problème à un problème de couplage de poids minimum et de cardinalité maximum dans les graphes bi-partis pondérés. Un exemple est montré dans la figure 2.5.

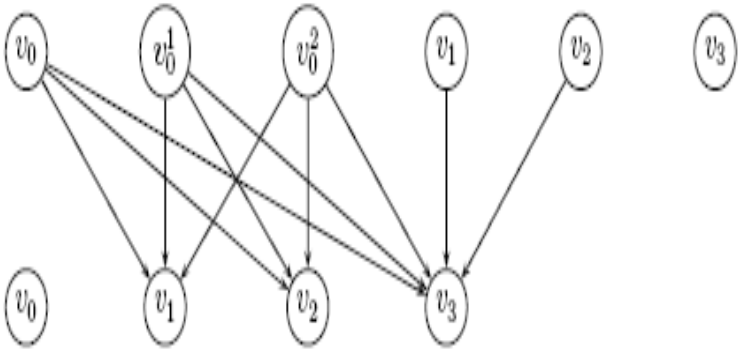

FIG. 2.5 – Le graphe bi parti associé

Le sommet v_0 est capable de répondre à v_1, v_2 et v_3. L'algorithme crée alors trois copies de v_0. Pour chaque aréte(v_i, q_j) on associe un poid qui corréspond au coût de réponse de la requête q_j en utilisant la vue v_i.

Donc, le problème de trouver l'ensemble M_1 est réduit à un problème de couplage de poids minimum et de cardinalité maximum dans le graphe bi-parti pondéré qui peut étre resolu en temps polynomiale.

La minimisation du poids assure d'une part que la somme du coût de traitement des

requêtes est minimum.

D'autre part la maximisation de cardinalité du couplage garantit la matérialisation du plus grand nombre des vues possibles.

Dans un deuxième temps, nous nous intéressons au choix d'un sous ensemble à partir de l'ensemble issu à l'aplication de la premiére phase de l'algorithme, tenant compte des conditions du problème.

2.6.3 L'algorithme intégré :

Cet algorithme a été réalisé par **Liang** et **Wang** et **Orlowska** en 2001 [3].

Le coût de traitemenet est supposé trés élevé lorsqu'aucune vue n'est matérialisée, cet algorithme essaye de réduire ce coût de traitement par la matérialisation de vues une par une, tantque le seuil maximum de la maintenance n'est pas dépassé.

Soit M l'ensemble de vues sélectionnées pour la matérialisation, et $U(M)$ le coût de maintenance de ces vues.

Soit $\tau(G, M)$ le coût de traitement pour répondre à toutes les requêtes en présence de l'ensemble à matérialiser M.

On considére la vue $v \in V(G) - M$, si celle ci doit être matérialisée dans la prochaine étape , le coût de maintenance changera avec une différence de $\delta T_v = U(M \cup \{v\}) - U(M)$ et le coût de traitement diminuera de $\tau(G, M) - \tau(G, M \cup \{v\})$ unité de temps à chaque étape.

L'algorithme sélectionne la vue $v \notin M$ dont le bénéfice $g(v)$ est maximum avec :

$$g(v) = \frac{\tau(G, M) - \tau(G, M \cup v)}{\delta T_v}.$$

Brievement, dans l'algorithme integré, on commence par selectionner une vue v_0 dont le bénéfice est maximal en l'absence des autres vues.

Cette vue est la premiére vue selectionnée dans l'ensemble M. En deuxième étape, pour chaque itération l'algorithme sélectionne la vue de bénéfice maximal en présence des vues matérialisées choisies précédèment.

La selection des vues de chaque itération est indépendante des séléctions déja entamées.

Algorithm 3 Algorithme intégré

Entrées: Un graphe $G = (V, E)$, contrainte de maintenance S ;

Sorties: Ensemble de vues à materialiser ;

-$M \leftarrow \emptyset$;

-soit v_0 la vue avec bénéfice $g(v_0)$ maximal ;

-$M \leftarrow v_0$;

-$\delta S \leftarrow S - U(M)$;

Tantque $\delta S > 0$ **Faire**

-gain$\leftarrow 0$;

Pour chaque $v \in V(G) - M$ **Faire**

-$g(v) = (\tau(G, M) - \tau(G, M \cup v))/\delta T_v$;

Si $g(v) >$gain **Alors**

-gain $= g(v)$; $w = v$;

FinSi

Fin Pour

Si $S - \delta T_w > 0$ **Alors**

-$\delta S \leftarrow \delta S - \delta T_w$;

-$M \leftarrow M \cup w$;

FinSi

Fin tantque

-Récupérer M ;

Brievement, dans l'algorithme integré, on commence par selectionner la vue v_0 dont le bénéfice est maximum.
Pour chaque itération, on selectionne la vue de bénéfice maximal en présence des vues déja matérialisées.

2.7 Conclusion

Dans ce chapitre on s'est intéressé à la résolution du problème de maintenance de vues sous contrainte de maintenance, et nous avons cité quelques algorithmes connus dans la

littérature.

Dans le chapitre suivant, nous allons voir d'autres méthodes de résolution basées sur le principe de colonie de fourmis connu dans la recherche opérationnelle.

Chapitre 3

Résolution par les colonies de fourmis

3.1 Introduction

Les algorithmes évolutionnaires jouent un rôle trés intéressant dans la résolution du probléme de matérialisation de vues dans un entrepôt de données. Les résultats éxperimentaux ont montrés que les algorithmes evolutionnaires ont donné des résultats plus proches aux solutions optimales. Parmi les algorithmes evolutionnaires connus nous trouvons les algorithmes génétiques et les colonies de fourmis.
Dans ce chapitre nous nous occupons des colonies de fourmis car elles sont les plus récentes.

3.2 Les méta-heuristiques

Il existe dans la recherche opérationnelle plusieurs classes de problèmes d'optimisation sériées selon des propriétés et des caractéristiques de difficulté et de complexité.
De tels problèmes sont trés fréquemment rencontrés dans la vie réelle, et dans certains cas les décideurs ont besoin des solutions immédiates. Les chercheurs se sont donc tournés vers d'autres techniques de résolution dites **heuristiques**. Ces heuristiques sont en fait des procédures intuitives propres à chaque problème, conçues étape par étape en utilisant le bon sens et la logique et s'orientant selon les préférences du décideur, de sorte à trouver une solution assez satisfaisante que possible.

En effet, les heuristiques, à la différence des méthodes exactes, sont souvent construites à partir de l'expérience, plutôt que d'une analyse scientifique qui prend en compte un maximum d'éléments et qui serait difficilement exploitable.

3.2.1 Les méta-heuristiques et leurs concepts

Au milieu des années 1970, sont apparues des méthodes qui supervisent l'évolution des solutions fournies par des heuristiques. Ces algorithmes ont été appelés **méta-heuristiques** et ont pour objectif de trouver des solutions dont la qualité est au-délà de ce qu'il aurait été possible de réaliser avec une simple heuristique. Quelques méta-heuristiques s'appuient sur des mécanismes de transition probabiliste et aléatoire. Cette caractéristique indique que plusieurs exécutions successives de ces méthodes peuvent conduire à des résultats différents pour une même configuration initiale d'un problème d'optimisation.
Les méta-heuristiques ont une grande capacité de trouver l'optimum global du problème. Contrairement à la plupart des méthodes déterministes, elles nécessitent ni point de départ, ni connaissace de gradient de la fonction objectif pour atteindre la solution optimal. Cependant elles demandent un nombre important d'évaluations avant d'arriver à la solution du problème.
Les méta-heuristiques sont classées en deux groupes :

Les méta-heuristiques à solution unique :

Les méthodes itératives à solution unique sont toutes basées sur un algorithme de recherche de voisinage qui commence avec une solution initiale, puis l'améliorer pas à pas en choisissant une nouvelle solution dans son voisinage.
Nous donnons comme exemple : les méthodes de descente, le recuit simulé et la recherche tabou.

Les méta-heuristiques à population de solutions

Les méthodes d'optimisation à population de solutions améliorent au fur et à mésure des itérations, une population comme facteur de diversité.
Nous donnons comme exemple : La recherche par dispersion, les algorithmes génétiques et les colonies de fourmis , qui sont l'approche de notre résolution.

3.2.2 Algorithme de colonies de fourmis

Les algorithmes de colonies de fourmis sont des algorithmes inspirés du comportement des fourmis et qui constituent une famille de métaheuristiques d'optimisation.
Initialement proposé par **Marco Dorigo** et **al** dans les années **90**, pour la recherche de chemins optimaux dans un graphe, le premier algorithme s'inspire du comportement des fourmis recherchant un chemin entre leur colonie et une source de nourriture. L'idée originale s'est depuis diversifiée pour résoudre une classe plus large de problèmes et plusieurs algorithmes ont vu le jour, s'inspirant de divers aspects du comportement des fourmis.
En anglais, le terme consacré à la principale classe d'algorithme est *Ant Colony Optimization* (abrégé **ACO**). Il existe cependant plusieurs familles de méthodes s'inspirant du comportement des fourmis. En français, ces différentes approches sont regroupées sous les termes : algorithmes de colonies de fourmis, optimisation par colonies de fourmis, fourmis artificielles, ou diverses combinaisons de ces variantes.
L'idée originale provient de l'observation de l'exploitation des ressources alimentaires chez les fourmis. En effet, celles-ci, bien qu'ayant individuellement des capacités cognitives limitées, sont capables collectivement de trouver le chemin le plus court entre une source de nourriture et leur nid.
La premiére fourmi trouve la source de nourriture, via un chemin quelconque, puis revient au nid en laissant une piste de phéromone .
Les fourmis empruntent indifféremment tout les chemins possibles, mais le renforcement de la piste rend plus attractif le chemin le plus court.
Les fourmis empruntent le chemin le plus court, les portions longues des autres chemins perdent leur piste de pheromones.

Des biologistes ont ainsi observé, dans une série d'expériences menées à partir de **1989**, qu'une colonie de fourmis ayant le choix entre deux chemins d'inégale longueur menant à une source de nourriture avait tendance à utiliser le chemin le plus court. Un modéle expliquant ce comportement est le suivant :

- Une fourmi appelée **éclaireuse** parcourt plus ou moins au hasard l'environnement autour de la colonie ;
- si celle-ci découvre une source de nourriture, elle rentre plus ou moins directement au nid, en laissant sur son chemin une piste de phéromones ;
- ces phéromones étant attractives, les fourmis passant à proximité vont avoir tendance à suivre, de façon plus ou moins directe, cette piste ;
- en revenant au nid, ces mêmes fourmis vont renforcer la piste ;
- si plusieures pistes sont possibles pour atteindre la même source de nourriture, celle étant la plus courte sera, dans le même temps, parcourue par plus de fourmis que la longue piste car le plus court chemin comporte plus de phéromones. Cette quantité supérieure de phéromone incite plus de fourmis à choisir ce chemin plus court ;
- la piste courte sera donc de plus en plus renforcée, et donc de plus en plus attractive ;
- Les longues pistes finiront par disparaitre car les pheromones sont volatiles ;
- â terme, l'ensemble des fourmi a donc déterminé la piste la plus courte.

La figure 3.1 représente bien notre explication. La première fourmi trouve la source de nourriture (**F**), via un chemin quelconque (**a**), puis revient au nid (**N**) en laissant derrière elle une piste de phéromone (**b**).

Les fourmis empruntent indifféremment les quatre chemins possibles, mais le renforcement de la piste rend plus attractif le chemin le plus court.

les fourmis empruntent le chemin le plus court, les portions longues des autres chemins perdent leur piste de phéromones.

Les algorithmes de colonies de fourmis sont beaucoup étudiés depuis quelques années, le problème du voyageurs de commerce a fait l'objet de la premiére implimentation de l'algorithme. Le passage de la métaphore à l'algorithme est dans ce cas realtivement facile à faire puisque le problème du voyageurs de commerce est bien connu et etudié.

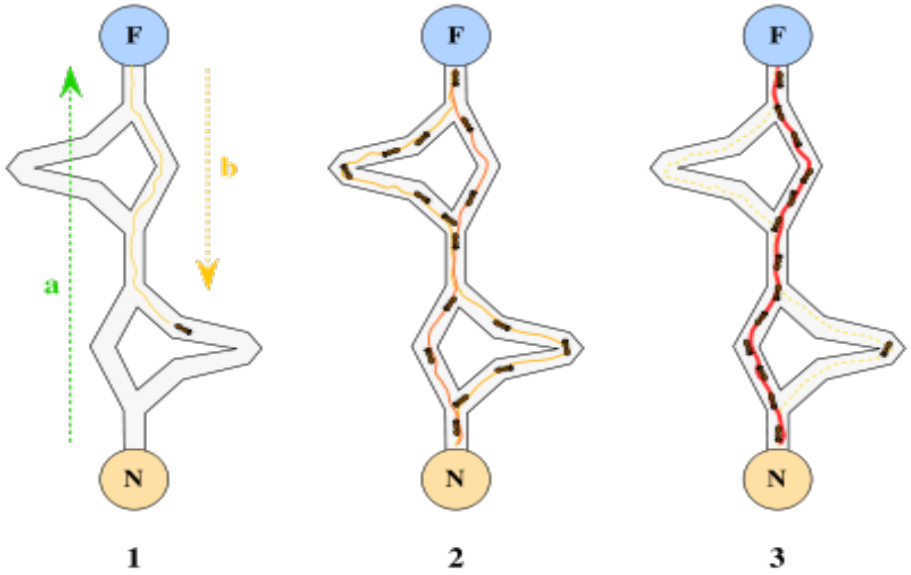

FIG. 3.1 – Le comportement des fourmis

3.3 Résolution du problème de materialisation de vues par l'algorithme de colonies de fourmis

Nous allons exposer l'algorithme de **A. Boukra** et **M. Ahmed Nacer** et **S. Bouroubi** nommé **Hybride** [1], aprés nous allons voir un autre algorithme qui utilise le principe de colonies de fourmis.

3.3.1 Algorithme Hybride(Bouroubi et all)

Soit $V = \{v_1, v_2, ..., v_L\}$ l'ensemble de vues et L le nombre de vues.
Notons qu'une solution $S \in \{0,1\}^L$ est écrite sous la forme suivante :
$S = (S_1, S_2, ..., S_L)$ avec :

$$S_i = \begin{cases} 1 & \text{si la vue } v_i \text{ est matérialisée} \\ 0 & \text{sinon} \end{cases}$$

La résolution repose sur la construction d'un graphe $G = (X, E)$ avec $|X| = L$ de la façon suivante :

– On associe à chaque vue $v_i \in V$ un sommet $x_i \in X$.
– Pour chaque couple (v_i, v_{i+1}) avec $i = 1, .., L-1$, on associe deux arcs dans G notés 0_i et 1_i de x_i vers x_{i+1}

Pour la construction d'une solution, la fourmi parcours tous les sommets de gauche à droite, et choisit soit l'arc 0_i ou 1_i pour aller du sommet x_i au sommet suivant x_{i+1}, le choix des arcs est effectué selon une probabilité de déplacement qui dépend des quantités de phéromone existantes dans les arcs.
Soit les deux arcs correpondants au sommet x_i, les quantités de phéromone sur les arcs 0_i et 1_i sont réspectivement τ_{0_i} et τ_{1_i}. A partir de ces deux valeurs nous pouvons définir une probabilité p_i pour générer un "1" à la position i, c'est a dire $S(i) = 1$.

$$p_i = \frac{\tau_{1_i}}{\tau_{1_i} + \tau_{0_i}}.$$

Initialement, chaque arc 1_i et 0_i avec $i = \{1, 2, ..., L-1\}$ a une quantité de phéromone positive fixe noté τ_0.

3.3.2 Mise à jour de la phéromone

Mise à jour globale

On utilise la meilleure solution obtenue aprés la génération de tous les individus notée S^{++}, donc la mise à jour de phéromone $\tau_{k_i}(k \in \{0,1\}, i \in \{1, ..., L\})$ est :

$$\tau_{k_i} \leftarrow (1-\rho).\tau_{k_i} + \rho.\Delta_{k_i}.$$

avec $\rho \in [0,1]$ le coefficient d'evaporisation et Δ_{k_i} la quantité de phéromone déposée telle que :

$$\Delta_{k_i} = \begin{cases} \frac{1}{1+c(S^{++})} & \text{si } S^{++} = k \\ 0 & \text{sinon} \end{cases}$$

avec $c(S^{++})$ est le coût de traitement de la meilleure solution S^{++}.

Mise à jour locale

Si l'arc k_i est choisi, la quantité de phéromone sur cet arc changera comme suit :

$\tau_{k_i} \leftarrow \tau_{k_i} - \alpha(\tau_{k_i} - \tau_0)$.

avec $\alpha \in [0, 1]$ et τ_0 est la quantité de phéromone initiale.

3.3.3 L'algorithme

Algorithm 4 Algorithme Hybride

Entrées: Population de taille p, Vecteur de probabilités V, Matrice de phéromones τ, L le nombre de vues ;

Sorties: Ensemble de vues à matérialiser ;

Pour $i = 1$ à L **Faire**

-$p_i = 1$;

Fin Pour

Pour $i = 1$ à L **Faire**

Pour $j = 1$ à L **Faire**

-$\tau_{ij} = 0.5$;

Fin Pour

Fin Pour

-Générer une premiére population $G(0)$;

-t=0 ;

Répéter

-t+=1 ;

-Géneration d'une nouvelle population $G(t)$;

-Classification stochastique de $G(t-1)$ et $G(t)$;

-Retenir les p meilleurs solutions ;

-Mise à jour globale ;

-Mise à jour de vecteur de probabilités selon la nouvelle matrice de phéromones obtenue ;

Jusqu'à La fin de condition

Notons que la classification stochastique qui classifie les individus selon un paramétre donné nommé **paramètre de balance** pour avoir les premiers individus qui donnent les meilleurs solutions. La procédure de cette classification stochastique est donnée comme suit :

Algorithm 5 Classification stochastique

ENTRÉES: $\lambda = 2P$ individus, N = λ, paramètre de balance p_f ;

SORTIES: Ensemble de vues par ordre ;

Pour $i = 1$ à N **Faire**

Pour $j = 1$ à $\lambda - 1$ **Faire**

-Générer $u \in U(0, 1)$;

Si I_i et I_j sont realisables ou $(u < p_f)$ **Alors**

Si $f(I_j) < f(I_{j+1})$ **Alors**

-Permuter (I_j, I_{j+1}) ;

FinSi

Sinon

Si $\Phi(I_j) > \Phi(I_j)$ **Alors**

-Permuter (I_j, I_{j+1}) ;

FinSi

FinSi

Fin Pour

Si Pas de permutations **Alors**

Exit ;

FinSi

Fin Pour

Notons que $\Phi(I_j)$ est le coût de maintenance associé à la solution (I_j) et $f(I_j)$ est le coût de traitement associé à la solution (I_j).

Concernant la procédure de génération, elle est comme suit :

Algorithm 6 La génération

Entrées: $q_0, \alpha, k, ;$

Pour $j = 1$ à p **Faire**

Pour $i = 1$ à L **Faire**

-Générer $u \in [0, 1]$.

Si $u < 1 - q_0$ **Alors**

-Générer $w \in [0, 1]$.

Si $w < p_i$ **Alors**

-Le bit i de S_j prend la valeur 1 ;

Sinon

-Le bit i de S_j prend la valeur 0 ;

FinSi

-Mise à jour locale ;

Sinon

Si $p_i > 0.5$ **Alors**

-Le bit i de S_j prend la valeur 1 ;

Sinon

-Le bit i de S_j prend la valeur 0 ;

-Mise à jour locale ;

FinSi

FinSi

Fin Pour

Fin Pour

3.3.4 L'algorithme CFGC

Dans notre modélisation graphique nous allons construire le graphe orienté complet $G = (V, E)$ telque $|V|$ est le nombre de vues.

Notons par :

J_i^k : L'ensemble de vues non encore visitées par la fourmi k quand elle est sur la vue v_i,

M : l'ensemble de vues déja materialisées.

c_j : le coût de traitement de $M \cup \{j\}$,

τ_{ij} : la quantité de phéromones déposées sur l'arc $v_i v_j$.

La régle de deplacement

Soit une fourmi k dans une itération donnée t existante sur la vue v_i. La fourmi doit aller à une autre vue pas encore visité.

La régle de déplacement dépend d'une probabilité appelée **probabilité de transition** notée p_{ij} telque :

$$p_{ij}^{k(t)} = \begin{cases} \frac{(\tau_{ij}(t))^{\alpha}(c_j)^{\beta}}{\sum_{l \in J_i^k}(\tau_{ij}(t))^{\alpha}(c_j)^{\beta}} & \text{si } j \in J_i^k, \\ 0 & \text{sinon .} \end{cases}$$

Avec $\alpha \in [0,1]$ et $\beta \in [0,1]$ deux paramètres de controle du coût de traitement et des quantités de phéromones déposées.

Mises à jour locale et globale

A l'itération t, et aprés la fin de trajet, la fourmi k laisse une certaine quantité de phéromone Δ_{ij} telle que :

$$d\Delta_{ij}^{k(t)} = \begin{cases} \frac{1}{1+L^k(t)} & (i,j) \in T^k(t), \\ 0 & \text{sinon.} \end{cases}$$

où $T^k(t)$ est le trajet parcouru (ensemble des vues choisies) par la fourmi k à l'iteration t et $L^k(t)$ est la longueur de ce trajet (coût de traitement associé à cet ensemble de vues selectionné).

L'expression de la mise à jour globale de la phéromones est donnée de la façon suivante :

$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t)$

Où $\Delta\tau_{ij}(t) = \sum_{i=1}^{m} \Delta\tau_{ij}^k(t)$. La mise à jour globale intervient lorsque toutes les fourmis effectuent leur tarjet dans une itération donnée t.

3.3.5 L'algorithme

Algorithm 7 Algorithme CFGC

Entrées: La matrice des coûts de traitements, m : le nombre des fourmis ; , V : le nombre de vues ;

Sorties: Ensemble de vues à materialiser ;

-Répartition aléatoire des fourmis sur les vues ;

-Initilisation de la matrice de phéromone avec $\tau_0 > 0$;

Tantque La condition est verifiée **Faire**

Pour i=1 a m **Faire**

Pour Chaque vue non visitée i **Faire**

Si la contrainte de coût de maintenance est verifiée selectionner la vue dans la liste J_i^k de vues restantes selon la probabilité de déplacement ;

Fin Pour

-Mise à jour locale ;

Fin Pour

-Mise à jour globale ;

Fin tantque

Remarques : Dans l'implimentation des algorithmes **Hybride** et **CFGC** la vue top (vue de dimension maximal entre toutes les vues existantes)sera toujours choisie, la génération sera donc effectuée pour les autres vues.

Dans notre implimentation la condition d'arrét est de boucler L fois.

les m fourmis sont réparties aléatoirement sur les V vues.

3.3.6 La complexité

Afin de se faire une idée, nous divisons l'algorithme en sections pour calculer la complexité :

1- Initialisation de matrice des phéromones et la répartition des fourmis sur les vues.

2- Déroulement d'une itération.

3- Mise à jour locale aprés la fin de l'itération.

4- Mise à jour globale.

5- Fin de la condition (Bouclage).

Procédons étape par étape :

1- On a une complexité $O(V^2 + m)$.

2- La complexité est $O(mV^2)$, puisque les opérations de calcul de la ville suivante nécessite un balayage de l'intégralité des villes.

3- La complexité est $O(mV^2)$.

4- La complexité est $O(V^2)$.

5- La complexité est $O(m)$.

Donc la complexité de cet algorithme est $O(V^2 + m + V.(m.(mV^2 + mV^2) + V^2))$.

La complexité de cet algorithme est donc $O(m^2.V^3)$

3.4 Conclusion

Nous avons exploité le principe de colonies de fourmis dans ce chapitre pour résoudre le problème de matérialisation de vues sous contrainte de maintenance.

La modélisation graphique de la dépendance entre les vues a facilité l'utilisation de l'algorithme **CFGC** en ajoutant des arcs au graphe de dépendance.

Le chapitre suivant utilise un nouveau concept pour la résolution de notre problème nommé le concept de la densité.

Chapitre 4

La matérialisation basée sur la densité

Dans ce chapitre nous allons présenter un nouveau algorithme basé sur la notion de la densité. A. Das et D. K. Bhattacharyya utilisent la fréquence d'accés des requêtes et le support de vues, pour regrouper les vues en classes en premier temps. Ensuite ils choisissent l'ensemble de vues à matérialiser selon certaines propriétés prédéfinies [5].

4.1 La notion de la densité

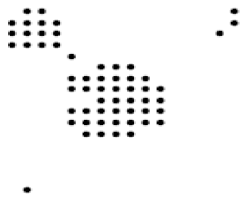

FIG. 4.1 – Echantillon 1

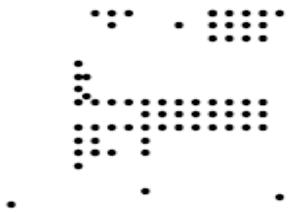

FIG. 4.2 – Echantillon 2

En voyant l'ensemble des points figurés dans les figures 4.1 et 4.2, nous pourrons facilement et d'une maniére évidente détecter l'ensemble de clusters des points et les points isolés .

La raison principale pour déterminer un cluster est qu'à l'intérieur de chaque cluster, on a

une densité typique de points qui est considérée plus élevée qu'à l'extérieur de ce cluster. Nous allons essayer d'exploiter cette notion intuitive du cluster dans un ensemble de vues d'un entrepôt de données, qui seront considéré comme un ensemble de points.
L'idée principale est que pour chaque point du cluster, le voisinage associé à un rayon donné doit y contenir au minimum un nombre fixe de points. Autrement dit, la densité de voisinage doit y dépasser un certain seuil.

4.2 La matérialisation de vues basée sur la densité

4.2.1 Utilisation du concept de la densité

Soit V un ensemble de vues. Pour former des clusters de vues et sélectionner l'ensemble de vues à matérialiser à partir de ces clusters. A. Das et D.K. Bhattacharyya ont réalisé un algorithme pour la matérialisation de vues en utilisant le concept de densité sous quelques hypothéses et sous un concept principal qui dit que le bénéfice de voisinage de chaque vue dans le cluster doit être une valeur prédéfinie. Le bénéfice de voisinage est une formule avec laquelle on peut mésurer la qualité du cluster.
Dans cet algorithme nous allons utiliser plusieurs sous concepts liés au concept de densité , donc il est trés important de les définir :

Définition. 1 *Le voisinage d'une vue v relatif à $MaxD$ noté par $N(v)$ est définie par :*
$N(v) = v \cup \{w \mid w \in children\ (v)\ et\ R(v) - R(w) \leq MaxD\}$
Avec $R(v)$ est la taille de la vue v.

Notons par $NMPV(v)$ la vue u à matérialiser parente de v, telle que la différence de taille entre u et v :($\delta R = R(u) - R(v)$) est la plus petite entre toutes les vues parentes de v materialisées. Nous dirons que la vue u est la plus proche vue matérialisée parente de v.

Définition. 2 *Une vue v est dite noyau si $benefit(N(v)) \geq MinBen$.*
Avec $MiBen$ le bénéfice minimum.

La détermination des paramètres $MinBen$ et $MaxD$ est une tâche dure et complexe, elle repose sur des notions et des caractéristiques non étudiées dans cette thése. Même les réalisateurs de cet algorithme ont considérés ces deux paramètres comme des données.
Dans le déroulement de l'algorithme, on associe à chaque vue un état fixe. Nous distinguons quatre catégories d'états pour les vues :

- Les vues classifiées qui sont déja associées à un cluster.
- Les vues non-classifiées qui ne sont pas encore associées à un cluster.
- Les vues bruits qui n'appartiennet à aucun cluster.
- Les vues maîtres (gérantes) qui sont des vues non classifiées et ayant leurs parents classifiées ou considérées comme des vues bruits.

4.2.2 L'algorithme de DVMA

Cet algorithme est basé sur la formation de cluster de vues. Il calcule le bénéfice de voisinage pendant la création du clusters.
Le bénéfice d'un voisinage noté $benefit(N(v))$ est donné par la formule suivante :
$benefit(N(v)) = [R(NMPV(v)) - R(v) \sum_{p \in N(v) \cup v} f_p] + \sum_{p \in N(v) \cup F} s(v)$.
F est l'ensemble de vues fréquentes, $R(v)$ la taille de la vue v, $f(v)$ est la fréquence d'accés de la vue v, $s(v)$ est le support ou encore la fréquence de la vue v, et enfin $R(NMPV(v))$ la plus proche vue parente de v.
Dans la pratique, et selon les résultats experimentaux de quelques expériences de matérialisation de vues, les chercheurs ont trouvé que les vues les plus fréquentes ont un grand rôle dans la performance des resultats, tel que la sélection de vues les plus fréquentes garantit avec un grand pourcentage la minimisation du coût de réponse total sur les requêtes, pour cette raison l'algorithme $DVMA$ introduit ces vues dans sa formule de bénéfice.
L'algorithme de DVMA est le suivant. L'algorithme complet sera ecrit en premier temps, suivi par la procedure appellée CreateCluster qui est utilissée pour la formation d'un cluster.

Algorithm 8 Algorithme DVMA

Entrées: Graphe représentatif de vues, V = l'ensemble de vues, La fréquence d'accés de vues, F, $MaxD$, $MiBen$;

Sorties: L'ensembles de vues à matérialiser ;

-Considérer toutes les vues de V comme de vues gérantes ;

-S=$\{v_1\}$ (v_1 la vue top) ;

-Temp="True" ;

-clid = obtenir un nouveau cluster id ;

Tantque il existe une vue gérante **Faire**

-Obtenir la gérante v de plus petite taille parmi les vues gérantes existantes et de dimension maximale ;

-Temp = CreateCluster(V, v, $clid$,$MaxD$, $MiBen$) ;

Si Temp ="True" **Alors**

-clid =obtenir un nouveau cluster id ;

FinSi

Fin tantque

La procédure CreateCluster est la suivante :

CreateCluster(V,v, $clid$, $MaxD$, $MiBen$)

Si $benefit(N(v)) < MinBen$ **Alors**

- v.noise = "True" ;

- **Return** False

FinSi

Sinon

- v.classifiée = "True" ;

- $S = S \cup v$;

- seeds = $\{w \mid w \in N(v)$ et w.classifié="False"$\}$;

Pour tout $s \in$ seeds mettre s.classisfié = "True" ;

TantQue seeds non vide **Faire**

Pour tout $s \in$ seeds

Si $benefit(N(v)) \geq MinBen$ Alors

$S = S \cup v$.

- Results = $\{w \mid w \in N(s)\}$;

Pour tout $r \in$ Results

Si r.classifié="False" **Alors**

-seeds=seeds $\cup r$; -r.classisfié = "True" ;

FinSi

FinPour

FinSi

-seeds=seeds-$\{s\}$;

FinPour

Fin Tant Que

-**Return** "True" ;

FinSi

Cet algorithme est construit sur quelques hypothéses :

1. les vues à materialiser sont sélectionnées indépendement.
2. Il n'existe aucune contrainte de ressource.
3. les vues sont organisées sous forme de représentation graphique entre les vues définies auparavant.

Cet algorithme cherche les vues noyaux pour la matérialisation, il matérialise toujours la vue top, pour garantir la réponse sur toutes les requêtes existantes.
Dans chaque itération l'algorithme cherche la vue gérante de plus petite taille avec dimension maximale, aprés il calcule le bénefice $benefit(N(v))$ de cette vue.

Si le bénefice est inférieur à $MinBen$ la vues sera isolée , sinon, le cluster commence à partir de v et tous les enfants de v non classifiés seront des vues candidates pour la matérialisation.
Une vue candidate sera prise en calculant son bénefice, Si c'est une vue noyau, tout les enfants de cette vue seront des vues candidates aussi pour la matérialisation. Sinon elle sera marquée comme classifiée.
Le processus continue jusqu'à ce que la liste devient vide. Comme ça un cluster est formé, et nous construirons les autres clusters avec la même procédure.

4.3 La matérialisation basée sur la densité sous la contrainte de la maintenance

Parmi les hypothèses de la matérialisation de vues dans le $DVMA$ est qu'aucune contrainte de ressources est considérée.
L'algorithme matérialise l'ensemble de toutes les vues noyaux trouvées. Les vues seront toutes stockées et maintenues.
Dans la pratique, le temps de maintenance de vues à un grand effet sur l'ensemble de vues matérialisées, et c'est cette contrainte qui pose beaucoup de problèmes par rapport aux autres contraintes comme le volume de stockage.
Dans cette section, on va voir comment matérialiser les vues noyaux issues du $DVMA$ si la contrainte de maintenance est prise en compte en utilisant un algorithme simple.

4.3.1 Intervention de la contrainte de maintenance

Soit M l'ensemble de vues noyaux issues de l'algorithme $DVMA$, nous pouvons sélectionner le sous ensemble E_M de M tel que E_M satisfait la contrainte de seuil maximum de maintenance.
Les procédures sont montrées dans l'algorithme suivant noté $MCDVMA$:

Algorithm 9 Algorithme MCDVMA

Entrées: Graphe représentatif de vues, V = l'ensemble de vues, La fréquence d'accés de vues, F, $MaxD$, $MiBen$, La contrainte de maintenance S ;

Sorties: L'ensembles de vues à matérialiser ;

-Appliquer $DVMA$ et soit M l'ensemble de vues trouvées ;

-$E_M \leftarrow s$ tel que s est la vue top ;

Tantque la contrainte de maintenance est verifiée **Faire**

-$E_M \leftarrow \{v_0\}$ tel que v_0 est la vue non encore séléctionnée qui donne un coût de traitement minimal ;

Fin tantque

-Retenir E_M qui est l'ensemble de vues à matérialiser ;

4.4 Conclusion

Dans ce chapitre nous avons utilisé l'algorithme basé sur la densité pour choisir l'ensemble de noyaux dans un premier temps, puis nous avons sélectionné l'ensemble de vues qui respecte la contrainte de maintenance en utilisant une méthode simple.
Jusqu'ici, nous avons exposé les différents algorithmes connus et proposés. Pour tester la performance de ces algorithmes nous devons passer aux études expérimentales.

Chapitre 5

Les résultats experimentaux

5.1 Introduction

Dans cette section, nous présentons quelques résultats des etudes experimentaux. Tout les algorithmes ont été implémentés en langage Java.
Les résultats donnent une idée générale sur la performance des algorithmes proposés.
Un exemple pratique est donné dans la deuxième partie de cette section pour comprendre l'utilité d'entrepôt de données.

5.2 Les données

On associe un graphe de dépendance entre les vues noté $(L, \leq)$, de taille $N = 32$ tel que N est le nombre de vues, où chaque noeud (vue) notée v est caractérisée par les trois poids cités dans le troisième chapitre (La taille de tables r_v, Fréquences d'accés f_v, Fréquences de mise à jour g_v). Il est de même pour les arcs, tel que chaque arc de v vers u est caractérisée par deux poids (Coût de traitement de u en utilisant v noté q_{uv}, coût de maintenance de la vue u en utilisant v noté m_{uv}).
Nous avons besoin aussi de la valeur du seuil maximum de maintenance noté S.
Soit une aréte de v vers u , nous supposons que le coût de maintenance de u en utilisant v est inferieur au coût de maintenance de v en utilisant u.
Nous supposons aussi que le coût de maintenance de vues est dépendant de leurs tailles.

De plus, nous supposons que q_{uv} est inferieur à r_v, et que $U_{(u,v)}$ est à peu prés $1/10$ de r_u.

5.3 L'algorithme CFGC

Dans cette section, nous evaluons la performance de l'algorithme **CFGC**. Nous comparons les résultats de cet algorithme avec l'algorithme **Hybride**.

5.3.1 Le coefficient d'evaporation

Fixons $\alpha = 0.5$ et $\beta = 0.5$. D'aprés le graphe, le coût de traitement est minimum pour

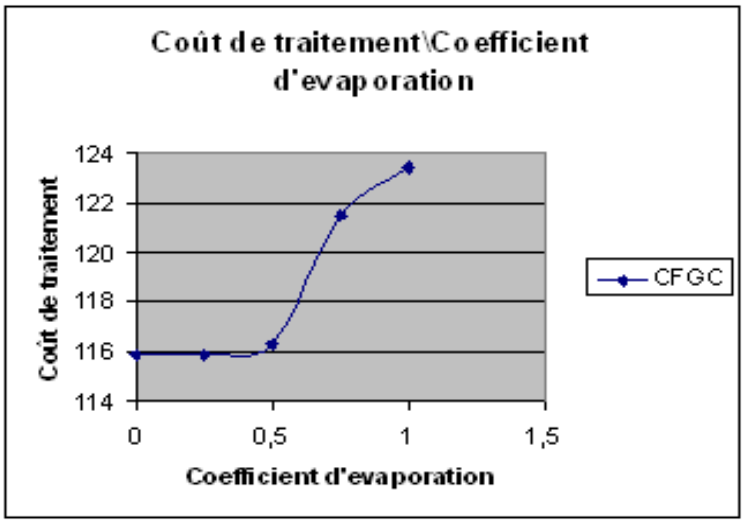

FIG. 5.1 – Coût de traitement en fonction de coefficient d'evaporation

$\rho = 0.25$ et $\rho = 0$.

5.3.2 Le seuil de maintenance

Fixons $\alpha = 0.5$ et $\beta = 0.5$ et $\rho = 0.25$ pour l'algorithme **CFGC**.
Pour l'algorithme **Hybride**, nous fixons le paramétre de mise à jour local $\alpha = 0.5$ et le coefficient d'evaporation $\rho = 0.25$ et le paramétre de balance $p_f = 0$.

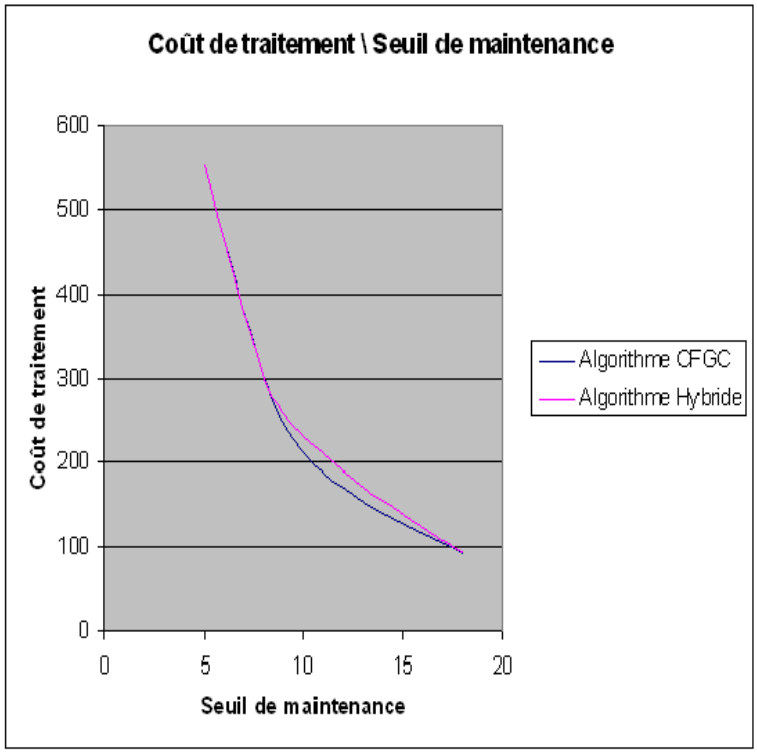

FIG. 5.2 – Coût de traitement en fonction de seuil de maintenance

La croissance du coût de maintenance entraîne une décroissance du coût de traitement. Pour chaque valeur du seuil de maintenance, le coût de traitement résultant de l'algorithme **CFGC** est inférieur à celui de l'algorithme **Hybride**.

5.3.3 Le nombre de vues

Le graphe représente le coût de traitement en fonction de nombre d'individus (vues) pour les algorithmes **CFGC** et **Hybride**. Pour une taille donnée, on a enregistré la meilleure solution obtenue pour chaque algorithme.

D'aprés le graphe, les coûts de traitements résultants de **CFGC** sont inférieurs au coût de traitements résultants de l'**Hybride** pour les tailles 32, 64, 128. Les coûts de traitements pour les tailles 4 et 8 sont égaux pour les deux algorithmes.

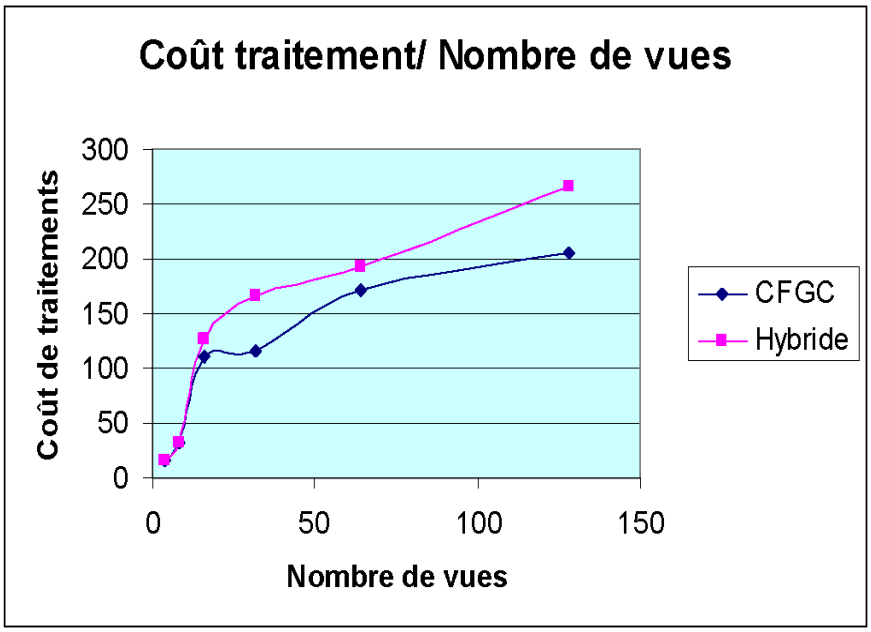

FIG. 5.3 – Coût de traitement en fonction du nombre de vues

5.4 L'algorithme MCDVMA

5.4.1 La solution

Cette section est consacrée à evaluer la performance de **MCDVMA**, ainsi qu'une étude comparative avec les algorithmes Intégré et de deux phases. Pour les différentes valeurs de $MaxD$ et $MinBen$, nous enregistrons les meilleurs solutions obtenues. L'algorithme affiche l'ensemble des clusters et l'ensemble de vues à matérialiser, ainsi que le coût de traitement total de cet ensemble(figure 5.4).

Dans cet exemple, les vues sont réparties en cinq clusters, $C1, C2, C3, C4, C5$. Les vues v_0 et v_{31} sont isolées.

La solution de cet ensemble est 115.89.

On trouve dans la colonne Matérialisation, les vues à matérialiser marquées par "Oui", et les vues non concérnées par la matérialisation marquées par "Non".

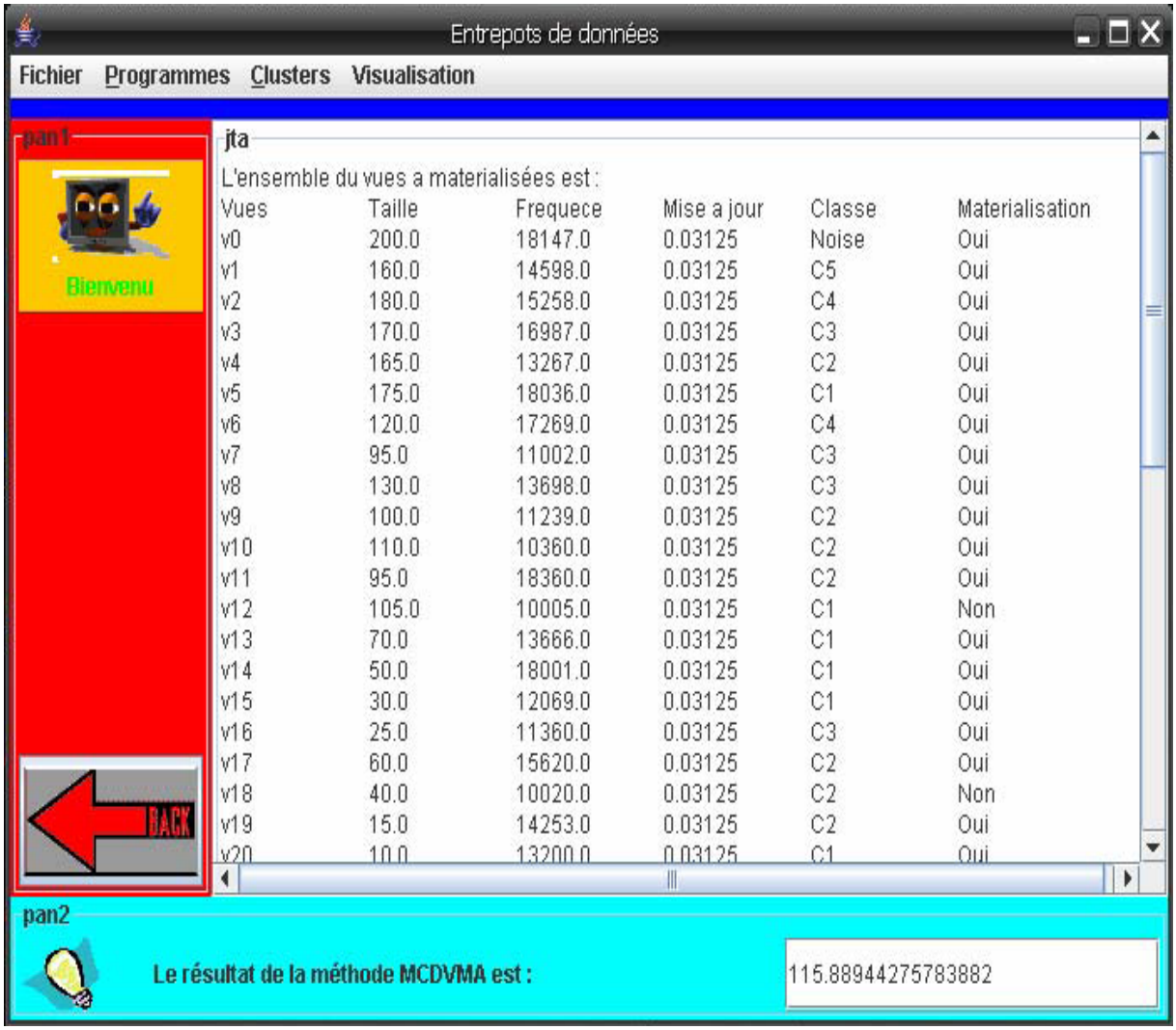

FIG. 5.4 – Résultats de l'application de MCDVMA

5.4.2 Le nombre de vues

D'aprés le graphe, les résultats obtenus par l'application de l'algorithme **MCDVMA** sont meilleurs que ceux de l'**Intégré** et de **De deux phases**.

La comparaison entre ce graphe et celui de la figure 5.3 indique que le coût de traitement est minimal pour l'algorithme **CFGC**. Le coût de traitement résultant de l'algorithme **MCDVMA** est inférieur à celui de l'**Hybride**.

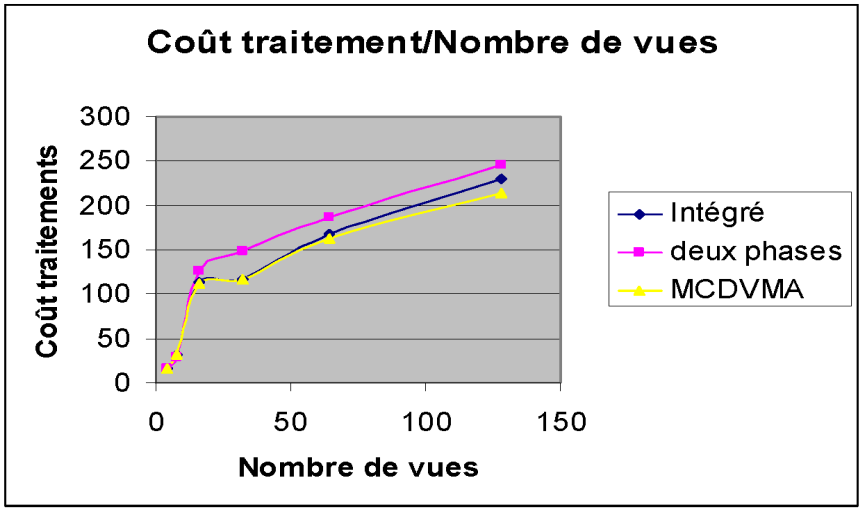

FIG. 5.5 – Coût de traitement en fonction du nombre de vues

5.4.3 Le seuil de maintenance

D'aprés le graphe, les résultats obtenus par l'application de l'algorithme **CFGC** sont meilleurs que ceux de l'**Hybride** et de **MCDVMA**.

La comparaison entre les résultats de **MCDVMA** et ceux de l'algorithme **Hybride** montre que les coûts de traitements du premièr sont meilleures que le second pour tout seuil de maintenance.

Par evidence, La croissance du coût de maintenance entraîne une décroissance du coût de traitement.

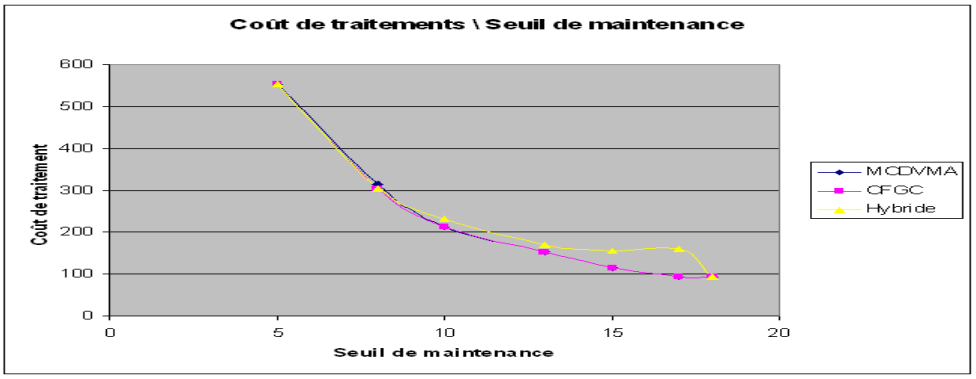

FIG. 5.6 – Coût de traitement en fonction de seuil de maintenance

5.5 Exemple pratique

Une grande entreprise veut rassembler toutes les nuits dans un entrepôt de données des informations sur les ventes du jour afin de dresser des tableaux de bord sur les ventes. L'entreprise dispose d'un système d'information complexe, constitué des éléments suivants :

- des applications et bases de données éparses et hétérogènes sur les produits qu'elle vend,
- des applications et BD, également variées, sur les clients,
- idem sur les personnels de l'entreprise.

L'entrepôt de données à modéliser doit pouvoir fournir le chiffre d'affaires des ventes d'un produit, par date, client, et vendeur, ainsi que toutes les sommations possibles de chiffre d'affaires.

Une vente correspond à un produit et un seul.

Les produits sont regroupés par famille de produits.

La vente est effectuée par l'un des vendeurs du service de vente spécialisé dans le produit.

La semaine de vente est le numéro de semaine dans l'année.

5.5.1 Modélisation de base

La table de faits de l'entrepôt de données sera alors la suivante :

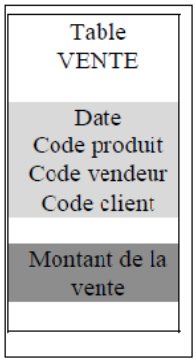

FIG. 5.7 – Table vente

En grisé clair, apparaît la clé multiple de l'enregistrement, constitué de 4 éléments : date, code produit, code vendeur, code client.
En grisé foncé, figure la variable à mesurer, appelée mesure : montant de la vente.
Cette table VENTE pourrait suffire à faire des sommations de chiffre d'affaires, si l'on se contentait des codes sur les éditions. C'est la table fondamentale de l'entrepôt.
En fait, la plupart du temps, chacun des éléments de la clé multiple de la table de faits renvoie à un certain nombre d'attributs. Ici, par exemple, le code produit sera utilement complété par :

- Libellé du produit.
- Code famille de produit.
- Libellé famille.
- Nombreuses informations complémentaires possibles.

On fait alors une jointure entre la table de faits et une table de dimension PRODUIT, selon le schéma suivant :

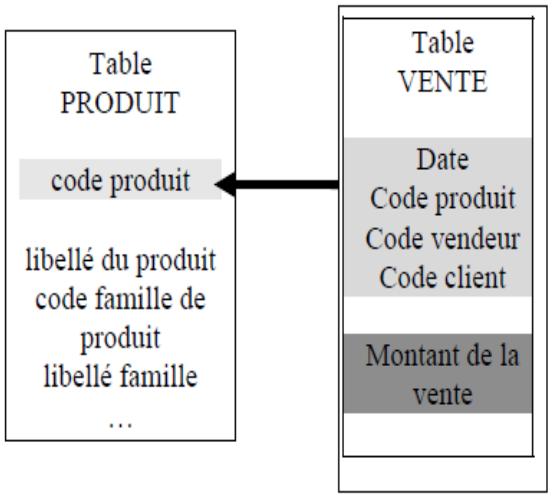

FIG. 5.8 – Table de faits (VENTE)+ table de dimension(PRODUIT)

La table de dimension PRODUIT n'a pas de mesure, elle comporte seulement des attributs du produit. La clé « code produit » correspond à l'élément « code produit » de la clé multiple de VENTE.
De la même manière, les autres éléments de la clé multiple renvoient en général chacun à

une table de dimension, selon le schéma suivant :

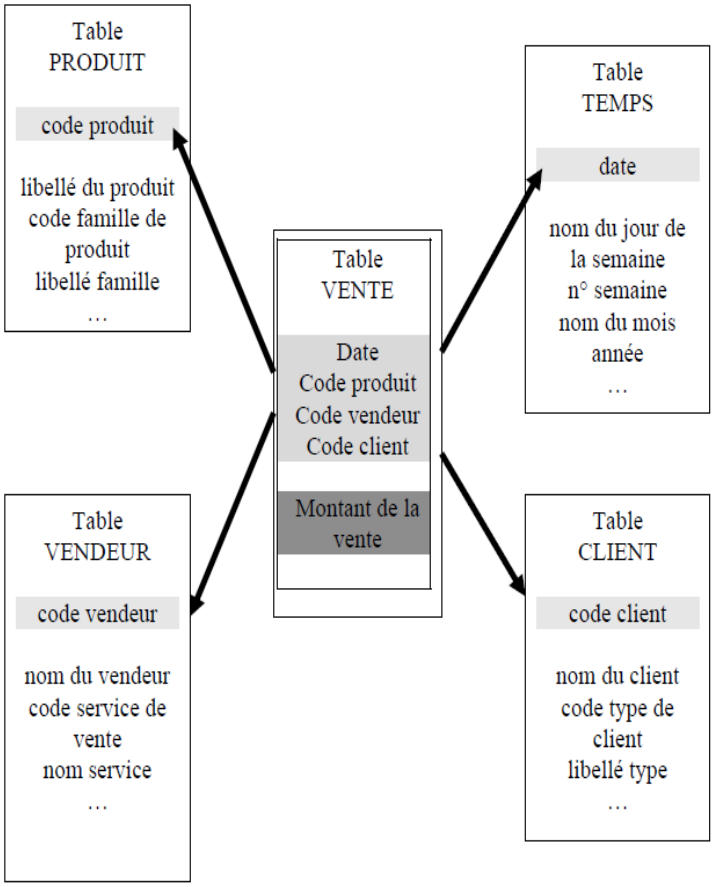

FIG. 5.9 – Table de faits (VENTE)+ les tables de dimension

Ce schéma, avec au centre la table de faits, et autour les tables de dimension jointes, s'appelle schéma en étoile.

Ce schéma est caractéristique de la modélisation dimensionnelle (du nom des dimensions) la plupart du temps mise en oeuvre dans la conception d'un entrepôt.

Un schéma en étoile peut également être représenté sous forme de cube de données.

A partir de la base de données relationnelle figurée par notre schéma en étoile, il est possible de développer un logiciel simple (à base de SQL par exemple) qui édite des sommes de « montant de la vente », ou chiffres d'affaires.

Dans le tableau ci-dessous, les éléments sont les totaux des montants vendus toutes dates et clients confondus.

Vendeur / produit				Produit : ski		
Vendeur : Dupont				10500		

FIG. 5.10 – Tableau a deux dimensions

Le vendeur Dupont a vendu pour 10500 euros de skis sur la période considérée.
Ce tableau peut également être produit pour une date donnée (sélection sur la clé date). L'empilement de ces tableaux par date peut être figuré par le cube ci-dessous vu en perspective :

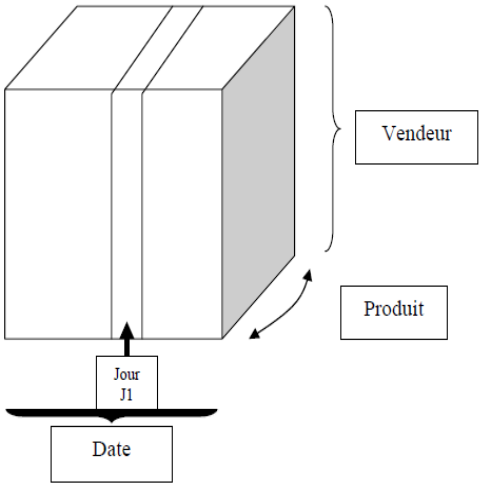

FIG. 5.11 – Cube de données à 3 dimensions

Lorsque le nombre de dimensions est de 3, la base de données peut être représentée par un cube. Lorsqu'il est supérieur à 3, comme c'est le cas dans l'exemple (qui comporte 4 dimensions), c'est un hypercube. Pour simplifier, on parle dans tous les cas d'un cube de données.

5.5.2 Les coupes

Dans l'exemple traité, et représenté par le schéma en étoile(Figure 5.9), le cube de données est un hypercube à quatre dimensions : produit, client, vendeur, date.

Graphiquement, on peut dessiner en perspective quatre types de cubes à trois dimensions :

A. client, vendeur, date (pour chaque valeur de produit)

B. produit, vendeur, date (pour chaque valeur de client)

C. produit, client, date (pour chaque valeur de vendeur)

D. produit, client, vendeur (pour chaque valeur de date)

Dans chaque cube, l'élément de base est la mesure montant de la vente.

Tracé de cube D :

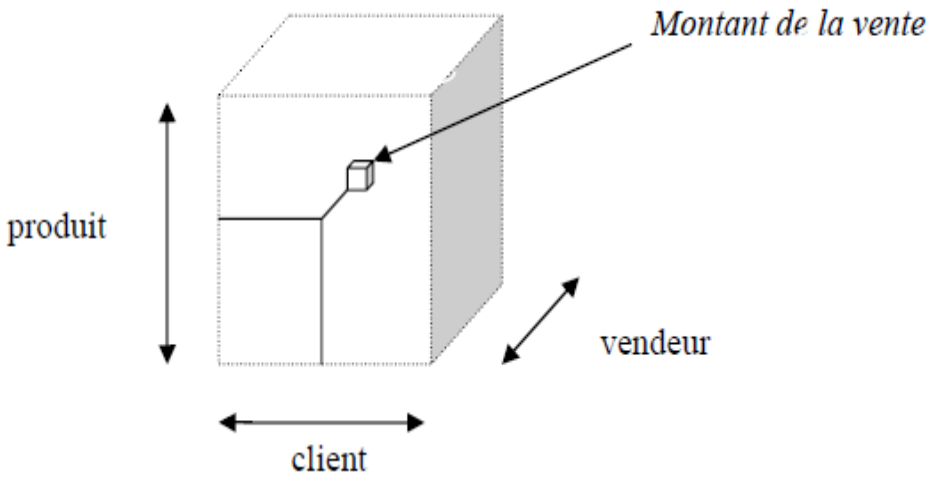

FIG. 5.12 – Un des types de cubes à trois dimensions

On peut tracer autant de cubes D qu'il y a de valeurs pour la variable date.

Un cube D représente une coupe de l'hypercube à 4 dimensions, selon une valeur de la variable date.

De même, on peut faire des coupes du cube D pour toutes les valeurs de produit, par exemple. On obtient alors autant de tableaux à deux dimensions (client, vendeur) qu'il y a de valeurs à produit.

A partir de D, on peut faire trois types de tableaux à deux dimensions : (client, vendeur),

(produit, vendeur), (client, produit).

A partir de l'ensemble A, B, C, D, on peut faire en plus les trois coupes qui gardent date (client, date), (produit, date), (vendeur, date), donc en tout six types de tableaux à deux dimensions.

5.5.3 Agrégats

Le but d'un entrepôt de données est la présentation de tableaux de bord.

On a compris dans ce qui précède que lorsque le nombre de dimensions du cube de données est n, avec $n > 2$, il faut faire des coupes en fixant les valeurs de $n - 2$ dimensions, pour se ramener à un tableau à deux dimensions, donc affichable.

Plutôt que de couper, on peut aussi agréger les données, c'est-à-dire présenter un tableau à deux dimensions en sommant les valeurs de certaines (voire toutes) des $n-2$ dimensions restantes.

C'est le cas du tableau vendeur – produit de la figure 5.10.

Lorsqu'on crée dans la base de données une table enregistrant ces sommations, on parle de table d'agrégat.

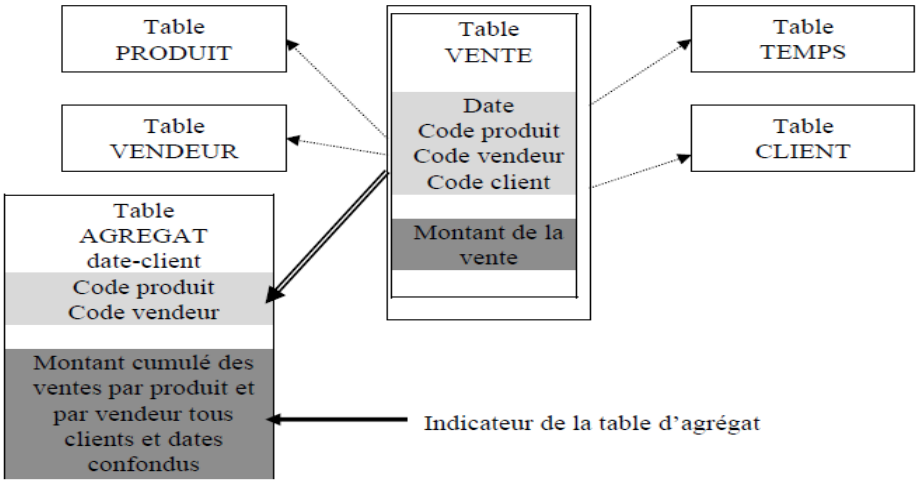

FIG. 5.13 – Création d'une table d'agrégat date-client

La création de tables d'agrégats a pour inconvénients :

1. Sur le plan conceptuel, de compliquer le modèle dimensionnel.
2. sur le plan technique, de multiplier l'espace de stockage sur disque.

En revanche, le résultat d'une requête d'un cumul est obtenu plus rapidement lorsque la table d'agrégats existe.

5.5.4 Analyse multi-dimensionnelle

Reprenons la figure 5.10, où les éléments sont les totaux des montants vendus toutes dates et clients confondus, avec en lignes les vendeurs et en colonnes les produits.

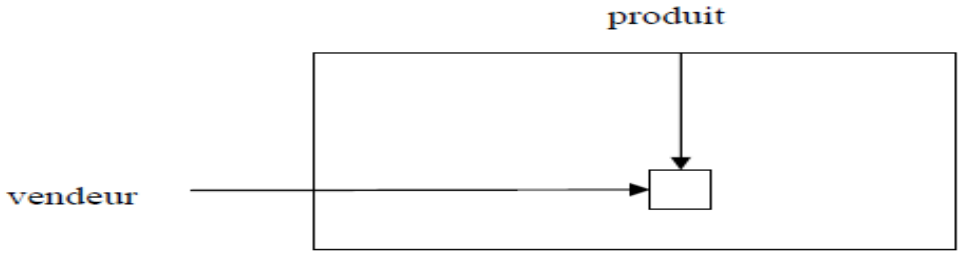

FIG. 5.14 – Tableau vendeur / produit

Il est indiqué dans l'énoncé qu'un produit appartient à une famille de produits, et un vendeur à un service de vente.
Il est donc légitime de vouloir éditer ce tableau en sommant les éléments par famille et/ou service.

Les dimensions vendeur et produit sont dites hiérarchiques, car elles peuvent se re-composer en service et famille respectivement.
Les variables service et famille, quant à elles, se décomposent.
Lorsqu'on va du tableau de la figure 5.14 au tableau A de la figure 5.15(respectivement B, C), on fait de l'analyse ascendante sur la dimension produit (respectivement vendeur, vendeur et produit).
Losqu'on part d'un des tableaux de la figure 5.15 pour éditer le tableau de la figure 5.14, on fait de l'analyse descendante.
L'analyse ascendante/descendante est appelée analyse multi-dimensionnelle.

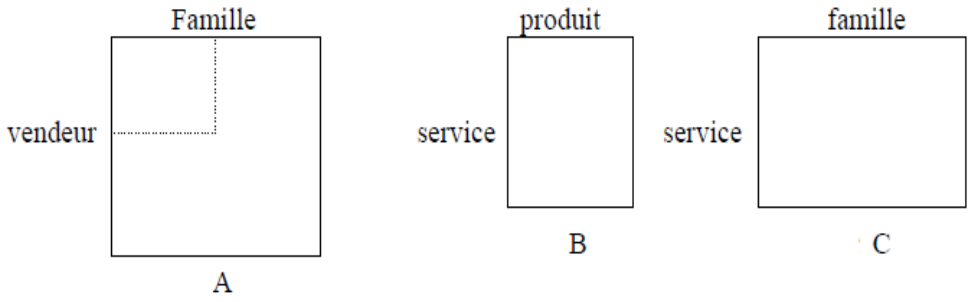

FIG. 5.15 – Sommations famille / service

Un cube D comme celui de la figure 5.12 peut donc représenter une multitude de possibilités de sommations, compte tenu des dimensions hiérarchiques et des possibilités d'agrégats :

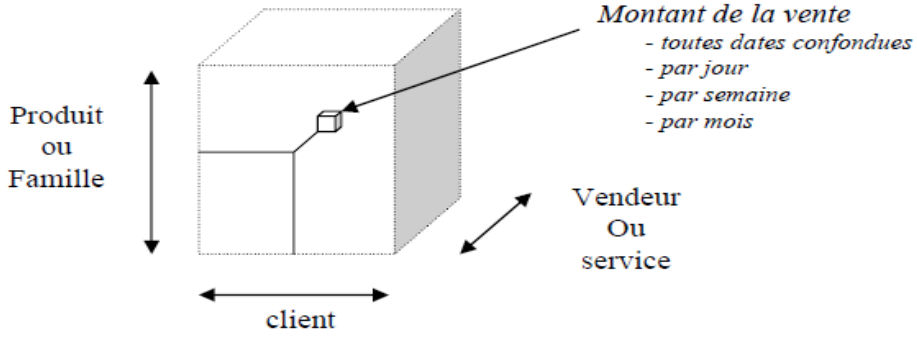

FIG. 5.16 – Autres possibilités de cubes D

5.5.5 Les vues et les requêtes

Une vue est une requête nommée. Une vue matérialisée est une table contenant les résultats d'une requête. Les vues améliorent l'exécution des requêtes en précalculant les opérations les plus coûteuses comme la jointure et l'agrégation, et en stockant leurs résultats dans la base. En conséquence, certaines requêtes nécessitent seulement l'accés aux vues matérialisées et sont ainsi exécutées plus rapidement.

Comme on a vu dans les parties conscrées aux agrégats et coupes, nous pouvons tracer toutes les tables possibles, par exemple le calcul d'agrégats a permi de tracer le tableau de la figure 5.10 qui est une vue puisqu'il affiche la réponse de la requête qui demanade la valeur du montant total d'un produit donné acheté par un client donné.
Une vue est caractérisée par :
1- **Le coût initial** : qui est généralement le nombre de lignes existantes dans la table.
2- **La fréquence** :Le nombre de fois de répitition de cette vue.
3- **La fréquence de mise à jour** :Le nombre de mises à jour effectuée durant une période donnée (ici c'est 24 heures)
On peut répondre sur la requête qui utilise les dimensions (PRODUIT, CLIENT) en utilisant le cube (PRODUIT,CLIENT,VENDEUR) par le calcul d'agrégats. Si nous avons le montant de vente d'un produit P acheté par un client C au vendeur V, donc nous pouvons calculer le montant de vente de produit P par le client C quelque soit le vendeur. Le coût de traitement de la requête (PRODUIT, CLIENT) est lié a la table(PRODUIT,CLIENT,VENDEUR).
La méthode de calcul du coût de traitement d'une requête est une opération spécifique de l'entreprise. Elle se base sur des notions en dehors de notre sujet.
Le mise à jour d'une table est les changements et les modifications et les vérifications efféctuées sur cette table. Il faut mettre à jour l'entrepôt de données suite aux changements apparus au niveau des sources, par exemple, si nous avons un nouveau produit il faut l'ajouter dans les base de données ou si un client a abondonné l'entreprise il faut le supprimer de cette bases de données. Ces changements effectués sur l'entrepôt associent des coûts, appelés coûts de maintenance.
Pour maintenir la table (PRODUIT, CLIENT), il suffit parfois de maintenir la table (PRODUIT,CLIENT,VENDEUR) donc, le coût de maintenance de la table (PRODUIT, CLIENT) est lié au coût de maintenance de la table (PRODUIT,CLIENT,VENDEUR).

5.5.6 La materialisation sous contrainte de maintenance

Pour un grand nombre de clients et une grande quantité de produits, l'entreprise doit avoir un entrepôt de données éfficace qui donne des réponses aux requêtes rapidement.
Dans notre exemple, la périodicité est de 24 heures donc l'entreprise doit répondre à n'im-

porte quelle requête dans ce délai. L'entreprise exploite les résultats des tables existantes dans l'entrepôt pour répondre par exemple au montant total des ventes.
La vitesse de son système d'information est donc liée à l'ensemble des tables existantes dans son entrepôt de données (les vues matérialisées), si par exemple, on compare le montant de vente totale entre deux produits P_1 et P_2, on trouve le résultat rapidement si cettes vues (PRODUIT) sont déja matérialisées. Par contre si nous avons seulement la table qui représente les produits par client ou par vendeur le calcul prend un temps plus long que dans le premier cas.
Supposons que cette entreprise est une branche d'une grande sociéte internationale et qu'elle doit rendre un revue chaque jour pour analyser ses ventes quotidiennes. L'entreprise doit faire alors face à cette contrainte de temps et son seul souci sera de performer son entrepôt de données qui doit répondre à toutes les requêtes possibles dans une période donnée(la contrainte de maintenance). Donc il faut séléctionner l'ensemble de vues à matérialiser qui minimise le coût de traitement des requêtes sous cette contrante de maintenance en utilisant les nouveaux algorithmes de séléction présentés dans ce mémoire.

5.6 Conclusion

Les études éxperimentales présentées dans ce chapitre montrent que les algorithmes proposés ont donné de meilleurs résultats que les algorithmes existants.
On peut améliorer les résultats de l'algorithme **CFGC** en multipliant le nombre d'agents (Fourmis) et le nombre de tournées. L'exemple pratique donne une idée générale sur l'utilité d'entrepôt de données qui constitue un élément incourtournable dans les entreprises et dans toutes les domaines (Gestion, Medecine, Assurances, Banques...).

Conclusion Générale

La matérialisation de vues est parmi les techniques les plus utilisées dans la conception des entrepôts de données pour minimiser le coût de traitement des reqûetes de type OLAP.

A travers cette étude, nous avons d'une part, décrit le probléme des entrepôts de données, sa formulation mathématique et les algorithmes de sa résolution, connus dans la littérature. Nous nous sommes basés sur le cas où la contrainte de maintenance de vues est prise en considération.

D'autre part, nous avons proposé deux nouveaux algorithmes, le premier est basé sur le principe des colonies de fourmis connu dans la recherche opérationnelle, cet algorithme est nommé **CFGC**.

Le deuxième nommé **MCDVMA** est basé sur la notion de la densité, avec la prise en charge de la contrainte de maintenance.

Les résultats expérimentaux ont montrés à travers une étude comparative, l'efficacité des deux algorithmes proposés par rapport aux algorithmes existants.

Comme perspectives, nous proposons d'ameliorer l'algorithme **CFGC**, en utilisant d'autres mises à jour de phéromones ou une autre probabilité de déplacement.

Pour l'algorithme **MCDVMA** nous proposons de définir une autre fonction de bénéfice ou essayer de faire rentrer les paramètres de maintenance de vues dans la fonction de bénéfice et dans le voisinage de vues.

Bibliographie

[1] A. Boukra, M. Ahmed Nacer, S. Bouroubi : Selection of views to materialize in data warehouse A hybrid solution. USTHB, Alger. 2006.

[2] H. Gupta, I. S. Mumick : Selection of views to materialize uder a maintenace cost constraint. In proc. 7th Int. Conf. Database Theory, pp 456-470. 1999.

[3] W. Liang, H. Wang, M. Orlowska : Materialized view selection under the maintenance time constraint. In proc. Of DKE01. 2001.

[4] J Xu Yu, X. Yao, C. H. Choi, G. Gou : Mterialized view selection as constrained evolutionary optimization. IEEE transactions, man and cybernetics. 2003.

[5] A. Das, D. K. Battacharyya : Density-Based view materialization. Tezpur University. India. 2005.

[6] R. Kimball : The Data Warehouse Toolkit. New York : Wiley, USA. 1996.

[7] V. Harinarayan, A. Rajaraman, J. D. Ullman : Implimenting Data Cubes Efficiently. Stanford University, Stanford. 1996.

[8] C. H. Choi, J Xu Yu, G. Gou : What difference heuristics make : Maintenance cost view selecton revisited. In proc.Third Int. Conf. Web Age Information Management. 2002.

[9] M. Ester, H. P. Kriegel, J. Sander, X. Xu : A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. University of Munich. Germany. 1996.

[10] H. Uchiyama, K. Runapongsa, T. J. Teorey : A Progressive View Materialization Algorithm. University of Michigan, Ann Arbor. 1999.

[11] B. Ashadevi, Dr R. Balasubramanian : Optimized Cost Effective Approach for Selection Views in Data Warehousing. Jsc s T , vol 9 Num 1, April 2009.

[12] H. Gupta : Selection of views to materialize in data warehouse. In proc. 6th Int. Conf. Database Theory, pp 98-112. 1997.

[13] W. H. Inmon : Building the Datawarehouse. John Wiley and sons. 2002.

[14] G. K. Y. Chan, Q. Li, L. Feng : Optimized Design of Materialized Views in a Real-Life Data Warehousing Environment. International Journal of Technology, vol7, Numb 1. 2000.

[15] S. Chandhuri, U. Dayal : An OverView of Datawarehousing and OLAP Technology : SIGMOD Record, 26(1) : 65-74. 1997.

[16] M. Dorigo, V. Maniezzo, A. Colorni : The Ant System : Optimization by a colony of cooperating agents. IEEE transactions, man and cybernetics. Part B, Num 1, p : 1-13. 2003.