MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS – SETIF
UFAS (ALGERIE)

MEMOIRE

Présenté à la Faculté des Sciences de l'Ingénieur
Département de l'informatique
Pour l'Obtention du diplôme de

MAGISTER

ECOLE DOCTORALE D'INFORMATIQUE (STIC)

Option : Ingénierie des Systèmes Informatiques (ISI)

Par
Mr : DAÂS Abdelaziz

THEME

Optimisation des requêtes dans le datawerehouse

Soutenu le : ……..devant un Jury composé de :

Touahria Mohamed	M.C à l'université de Sétif	Président
Khababa Abdellah	M.C à l'université de Sétif	Rapporteur
Moussaoui Abdelouahab	M.C à l'université de Sétif	Examinateur

Remerciement

Je remercie dieu le tout puissant qui m'a donné
La volonté et la patience pour
Mené a bien mon modeste travail.

Je tient beaucoup a remercié
Mon encadreur Mr KHABABA d'avoir
Accepter d'encadrer ce travail et ainsi
Que toute l'aide précieuse apporter
A mon égard.

Je tient également à remercier tout les
Membres du jury, qui ont accepter
D'examiner ce travail.

Je remercie tout les enseignant qui
Nous on orienté pendant tout
Notre parcours.

Je remercie tous mes camarades
En post-graduation

Merci, à tous ce qui m'ont aidé

Merci.

Dédicace

Le présent mémoire est dédié

Mes parents et pour leurs encouragements et leurs prières.

Ma femme pour le soutien.

Mes frères et ma sœur.

Tous Mes amis de l'académie surtout

Nabil,Omar,Lazhar,Nacer,Saci

Table des matières

CHAPITRE II : La sélection des vues matérialisées

dans l'entrepôt de données

CHAPITRE III : Les algorithmes génétiques

<u>CHAPITRE IV</u> : La sélection des vues matérialisées par les A.Gs Multiobjectifs

Liste des figures

Liste des tableaux

Introduction générale.

Aujourd'hui dans le contexte économique concurrentiel, l'acquisition, l'analyse et l'exploitation de l'information sont devenues des conditions stratégiques et inévitables pour chaque entreprise. Ainsi, afin de garantir leur persistance et croissance, les entreprises sont obligées à l'avance, de profiter de l'expertise dans ce domaine.

Les entrepôts de données (Data Werehouse) ont émergé comme solution potentielle répondant aux besoins du stockage (l'expertise) et de l'analyse de grands volumes de données, et de pallier les insuffisances des systèmes transactionnels (OLTP On Line Transactionnel Processing).

L'analyse de données de l'entrepôt se fait par les outils (OLAP : "*On-Line Analytical Processing"* [6]). Les requêtes formulées –vues- par cet outil –OLAP- comportent généralement l'agrégation qui est réalisée par la clause GROUP-BY du langage d'interrogation structuré SQL (Structured Query Language).

L'agrégation de données présente un coût très important qui diminue les performances du système OLAP. Par conséquent, des techniques ont été développées pour ajuster (remédier) cette dégradation. Parmi ces techniques, on cite les vues matérialisées. Ces vues améliorent l'exécution des requêtes, en pré-calculant les opérations les plus coûteuses comme la jointure et l'agrégation, et en stockant leurs résultats dans la base. Contrairement à une vue classique qui est calculée à chaque consultation à partir de la base. Le résultat des requêtes OLAP peut être critique dans l'aide à la décision, leur information doit être la plus récente. Pour cette raison la maintenance de l'ensemble de ces vues devient une propriété importante. En contre parti, cette propriété perturbe le système.

La sélection de ces vues présente un problème ; la matérialisation de tous ces vues diminue au maximum le coût d'interrogation (requête) ce qui implique un coût de rafraîchissement (maintenance) élevé. Alors que les deux coûts doivent varier adéquatement. En conséquence, Ce problème devient un problème d'optimisation à des objectifs multiples liés aux systèmes et aux utilisateurs. Ce problème a été

considéré Non Polynomial-difficile [20], on n'espère pas le résoudre de manière exacte. Donc il est indispensable d'utiliser des méthodes métaheuristiques.

Ainsi, l'objectif de notre travail est d'étudier l'utilisation de ces méthodes dans la sélection des vues, en tenant en compte les coûts de requêtes et de maintenance. Après une étude bibliographique sur les travaux dans ce domaine, on a remarqué que la majorité des contributions n'ont pas utilisé les algorithmes génétiques ; comme méthode métaheuristique. Ces méthodes ont donnée de meilleurs résultats par rapport à d'autre dans des cas similaire à notre problème.

Comme solution proposée à ce problème, nous avons basée sur l'application des algorithmes génétiques multi-objectifs sur la sélection des vues à matérialiser en se fondant sur un modèle de coût approuvé [25].

Ce mémoire est organisé en quatre chapitres comme suit :

Le premier chapitre est consacré aux définitions des concepts fondamentaux relatifs aux systèmes décisionnels, tel que la modélisation multidimensionnelle, et aussi aux techniques d'optimisations des requêtes ; les index, la fragmentation, et les vues matérialisés. Dans notre étude on s'est focalisé sur le problème de sélection des vues.

Le deuxième chapitre est divisé en trois parties. La première expose en détail le contexte du problème de sélection des vues matérialisées, il s'agit des contraintes à prendre en considération dans la résolution du problème.

Dans la deuxième, nous rappelons les fondements mathématiques utilisés dans la résolution d'un problème. Enfin, dans la troisième partie nous donnons un aperçu sur les algorithmes et les approches développées dans la littérature, et comme les algorithmes génétiques sont les plus avantageux, nous les avons sélectionné dans notre étude comme technique d'optimisation.

Le troisième chapitre explique les algorithmes génétiques (AG), on a commencé par les notions de base des l'AGs. Ensuite les opérateurs génétiques tels que la sélection, le croisement et la mutation. Enfin, nous avons présenté une nouvelle technique développée pour les problèmes multi objectifs. Parmi les variantes proposées, La NSGA II (Nondominated Sorting Genetic Algorithm) qu'on a adopté dans notre étude.

Le Quatrième chapitre concerne une application de la méthode NSGA II sur la sélection des vues matérialisées.

1-Introduction

Avec la généralisation de l'informatique dans tous les secteurs d'activité, les entreprises produisent et manipulent de très importants volumes de données électroniques. Ces données sont stockées dans les systèmes opérationnels de l'entreprise au sein de bases de données de fichiers. L'exploitation de ces données dans un but d'analyse et de support à la prise de décision s'avère difficile et fastidieuse ; elle est réalisée le plus souvent de manière imparfaite par les décideurs grâce à des moyens classiques (requêtes SQL, vues, outils graphiques d'interrogation…).

Ces systèmes paraissent peu adapter pour servir de support à la prise de décision. Ces bases opérationnelles utilisent le modèle relationnel ; celui-ci convient bien aux applications gérant l'activité quotidienne de l'entreprise, mais s'avère inadapté au décisionnel. Face à cette inadéquation, les entreprises ont recours à des systèmes d'aide à la décision spécifiques, basés sur l'approche des entrepôts de données. Cependant, de tels systèmes restent difficiles à élaborer et sont souvent réalisés de manière empirique, rendant l'évolution du système décisionnel délicate. Actuellement, les entreprises ont besoin d'outils et de modèles pour la mise en place de systèmes décisionnels comportant des données évolutives.

L'objet de ce chapitre est de définir les concepts inhérents aux systèmes décisionnels.

- La section 2 définit les concepts fondamentaux relatifs aux systèmes décisionnels.
- La section 3 présente les techniques d'optimisations des requêtes tel que les index, la fragmentation, et en fin les vues matérialisés.

2- Les concepts de base :

Comme toute étude et recherche exige au préalable de définir les concepts de base. Alors, nous commençons en donnant la définition de l'entrepôt de données, ses caractéristiques, la modélisation multidimensionnelle.

2.1-Définition de l'entrepôt de données

Bill Inmon dans son ouvrage de référence « using the data Warehouse » [29] définit l'entrepôt de la façon suivante « l'entrepôt de données est une collection de

données orientés sujet, intégrées, non volatiles et historiées, organisées pour support d'un processus d'aide à la décision », dans la suite nous détaillons ces caractéristiques.

2.2- Caractéristiques des entrepôts de données

Les données d'un entrepôt de données possèdent les caractéristiques [67,68] :

- **orientées sujet :** Les données des entrepôts sont organisées par sujet plutôt que par application. Par exemple, une chaîne de magasins d'alimentation organise les données de son entrepôt par rapport aux ventes qui ont été réalisées par produit et par magasin, au cours d'un certain temps.
- **intégrées :** Les données provenant des différentes sources doivent être intégrées, avant leur stockage dans l'entrepôt de données. L'intégration (mise en correspondance des formats, par exemple), permet d'avoir une cohérence de l'information.
- **non volatiles :** A la différence des données opérationnelles, celles de l'entrepôt sont permanentes et ne peuvent pas être modifiées. Le rafraîchissement de l'entrepôt, consiste à ajouter de nouvelles données, sans modifier ou perdre celles qui existent.
- **historisées :** La prise en compte de l'évolution des données est essentielle pour la prise de décision qui, par exemple, utilise des techniques de prédiction en s'appuyant sur les évolutions passées pour prévoir les évolutions futures.

2.3- Systèmes OLTP versus systèmes OLAP

Les bases de données sont utilisées dans les entreprises pour gérer les importants volumes d'informations contenus dans leurs systèmes opérationnels. Ces données sont gérées selon des processus transactionnels en ligne (OLTP : "*On-Line Transactional Processing"* [6]) qui se caractérisent de la manière suivante [6] [30] [34] [4] :

- ils sont nombreux au sein d'une entreprise,
- ils concernent essentiellement la mise à jour des données,
- ils traitent un nombre d'enregistrements réduit,
- ils sont définis et exécutés par de nombreux utilisateurs.

L'exploitation de l'information contenue dans ces systèmes opérationnels est devenue une préoccupation essentielle pour les dirigeants des entreprises qui désirent améliorer leur prise de décision par une meilleure connaissance de leur propre activité, de celle de la concurrence, des employés, des clients et des fournisseurs. Les entreprises sont donc à la recherche de systèmes supportant efficacement les applications d'aide à la décision. Ces applications décisionnelles utilisent des processus d'analyse en ligne de données (OLAP : "*On-Line Analytical Processing"* [6]). Ces processus répondent aux besoins spécifiques des analyses d'information. Dans [6] E.F. Codd définit un cahier des charges comprenant douze règles que doivent satisfaire les systèmes décisionnels : l'analyse des données doit se faire de manière interactive et rapide, pour des données historiées quelconques. Ces processus OLAP se caractérisent de la manière suivante [6] [30] [34] [4] :

- ils sont peu nombreux, mais leurs données et traitements sont complexes.
- il s'agit uniquement de traitements semi-automatiques visant à interroger, visualiser et synthétiser les données.
- ils concernent un nombre d'enregistrements importants aux structures hétérogènes.
- ils sont définis et mis en oeuvre par un nombre réduit d'utilisateurs qui sont les décideurs. Le tableau 1.1 montre une comparaison entre le processus OLTP et OLAP

	Processus OLTP	**Processus OLAP**
Données	Exhaustives	Résumées
	Courantes	Historiques
	Dynamiques	Statiques
	Orientées applications	Orientées sujets (d'analyse)
Utilisateurs	Nombreux	Peu nombreux
	Variés(employés,directeurs)	Uniquement les décideurs
	Concurrents	Non concurrents
	Mises à jours et interrogations	Interrogations

	Requêtes prédéfinies	Requêtes imprévisibles et complexes
	Réponses immédiates	Réponses moins rapides
	Accès à peu d'information	Accès à de nombreuses informations

Tableau 1.1-Comparaison entre les caractéristiques des processus OLTP et OLAP[63]

2.4- Modélisation multidimensionnelle

La construction d'un modèle approprié pour un entrepôt de données nécessite de choisir, soit un schéma relationnel (le schéma en étoile, en flocon de neige ou en constellation) ; soit un schéma multidimensionnel.

La modélisation multidimensionnelle se base sur un sujet analysé considéré comme un point dans un espace à plusieurs dimensions. Les données sont organisées de manière à mettre en évidence le sujet (le fait) et les différentes perspectives de l'analyse (les dimensions). Le fait représente le sujet d'analyse. Il est composé d'un ensemble de mesures qui représentent les différentes valeurs de l'activité analysée. Par exemple, dans le fait Ventes (*cf.* Figure 1.1), nous pouvons avoir la mesure "Quantité de produits vendus par magasin". Une dimension modélise une perspective de l'analyse. Elle se compose de paramètres (ou attributs) qui servent à enregistrer les descriptions textuelles. Nous pouvons utiliser les paramètres textuels comme source des restrictions dans une requête. Par exemple, dans la requête "Quantité de produits vendus dans la région Sétif durant le mois de Janvier 2010". Nous trouvons les paramètres : région "Sétif " et mois "Janvier 2010". Une hiérarchie représente les paramètres d'une dimension selon leur niveau de granularité ou de détail. Les paramètres sont ordonnés par une relation "est_plus_fin" et notée P1 $\rightarrow$ P2.

2.4.1- Schémas relationnels

Il existe deux types de schémas relationnels. Les premiers sont des schémas qui répondent fort bien aux processus de type OLTP qui ont été décrits précédemment, alors que les deuxièmes, que nous appelons des schémas pour le décisionnel, ont pour

but de proposer des schémas adaptés pour des applications de type OLAP. Nous décrivons les différents types des schémas relationnels pour le décisionnel.

2.4.1.1- Le schéma en étoile

Dans ce schéma, il existe une relation pour les faits et plusieurs pour les différentes dimensions autour de la relation centrale. La relation de faits contient les différentes mesures et une clé étrangère pour faire référence à chacune de leurs dimensions. La figure 1.1 montre le schéma en étoile en décrivant les ventes réalisées dans les différents magasins de l'entreprise au cours d'un jour. Dans ce cas, nous avons une étoile centrale avec une table de faits appelée Ventes et autour leurs diverses dimensions : Temps, Produit et Magasin.

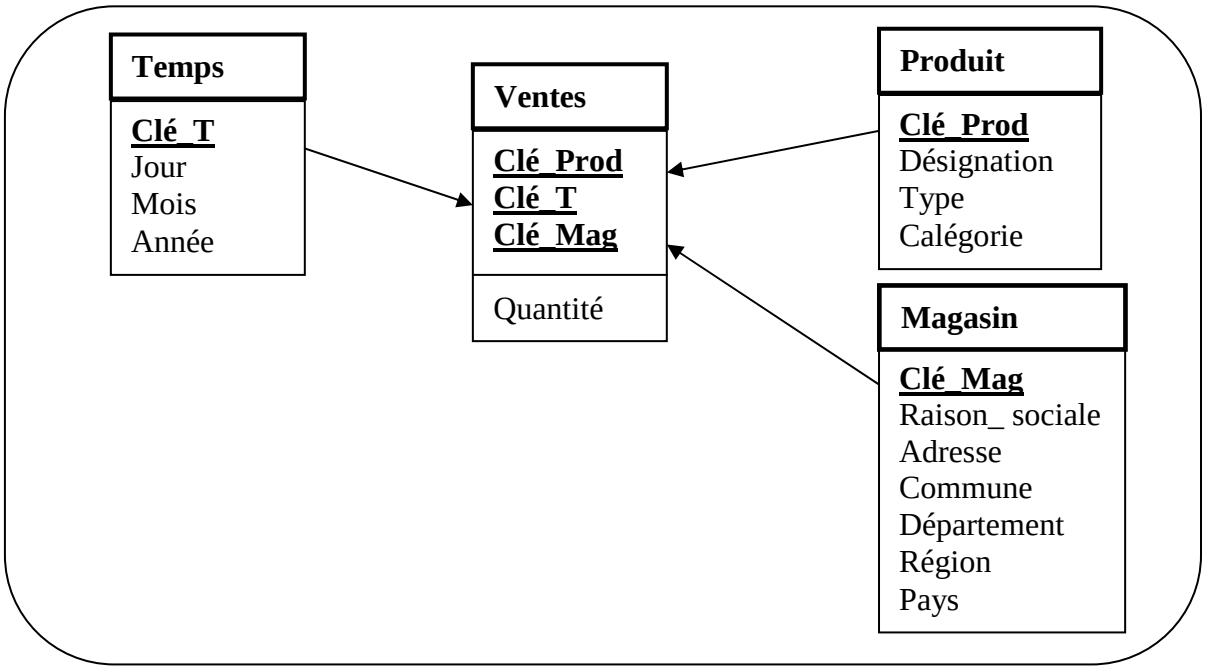


Figure 1.1 Exemple de modélisation en étoile

2.4.1.2- Le schéma en flocon de neige (Snowflake)

Il dérive du schéma précédent avec une relation centrale et autour d'elle les différentes dimensions, qui sont éclatées ou décomposées en sous hiérarchies. L'avantage du schéma en flocon de neige est de formaliser une hiérarchie au sein d'une dimension, ce qui peut faciliter l'analyse. Un autre avantage est représenté par la normalisation des dimensions, car nous réduisons leur taille. Néanmoins dans [34],

l'auteur démontre que la normalisation des relations des dimensions dans le but d'économiser l'espace disque est une perte de temps. Par contre, cette normalisation rend plus complexe la lisibilité et la gestion dans ce type de schéma. En effet, ce type de schéma augmente le nombre de jointures à réaliser dans l'exécution d'une requête. Les hiérarchies pour le schéma en flocon de neige de l'exemple de la figure 1.1 sont :

Dimension Temps = Jour ⟶Mois ⟶Année

Dimension Magasin = Commune ⟶Département⟶Région⟶ Pays

La figure 1.2 montre le schéma en flocon de neige avec les dimensions Temps et Magasin éclatées en sous hiérarchies.

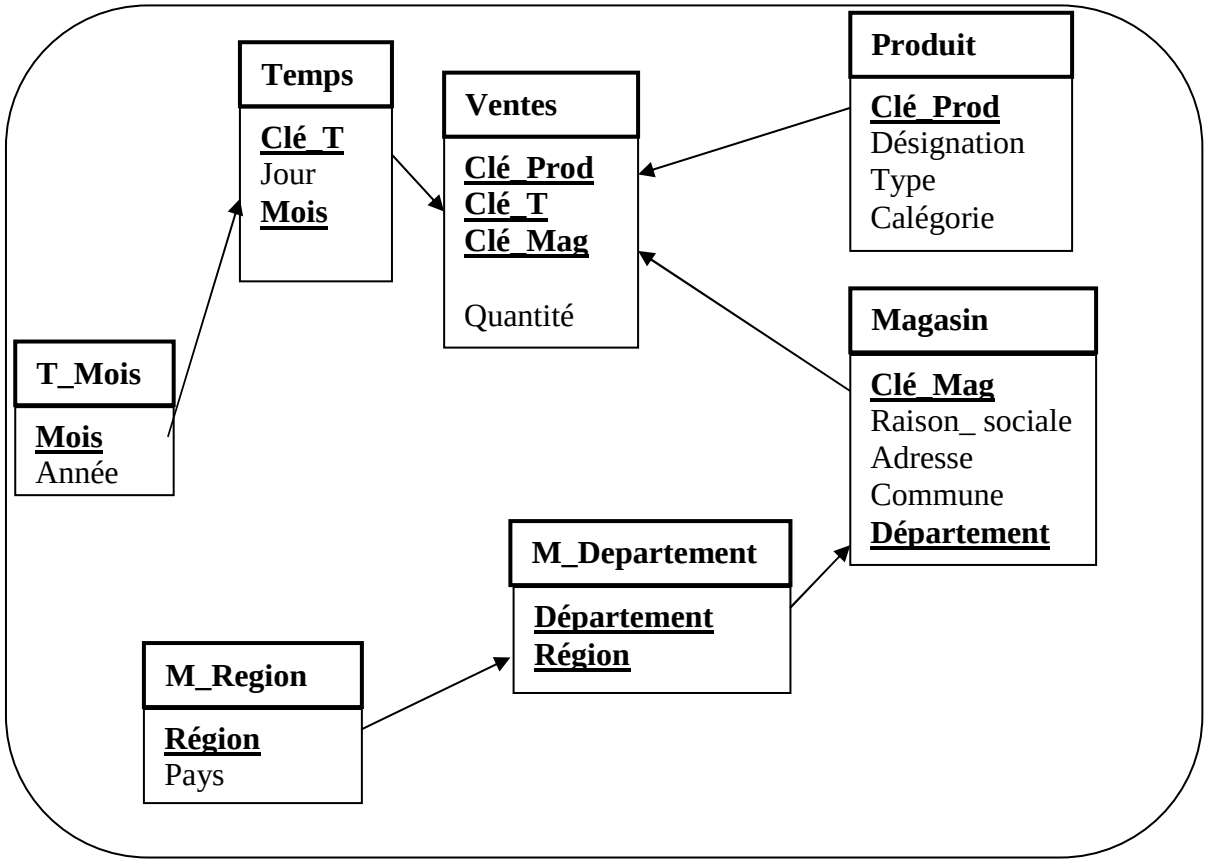


Figure 1.2 Exemple de modélisation en flocon de neige

Dans l'exemple ci-dessus, la dimension Temps a été éclatée en deux, Temps et T_Mois. La deuxième dimension Magasin, à été décomposée en trois : Magasin, M_Département et M_Région.

2.4.1.3- Le schéma en constellation

Le schéma en constellation représente plusieurs relations de faits qui partagent des dimensions communes. Ces différentes relations de faits composent une famille

qui partage les dimensions mais où chaque relation de faits a ses propres dimensions [3]. La figure 1.3 montre le schéma en constellation qui est composé de deux relations de faits. La première s'appelle Ventes et enregistre les quantités de produits qui ont été vendus dans les différents magasins pendant un certain jour. La deuxième relation gère les différents produits achetés aux fournisseurs pendant un certain temps.

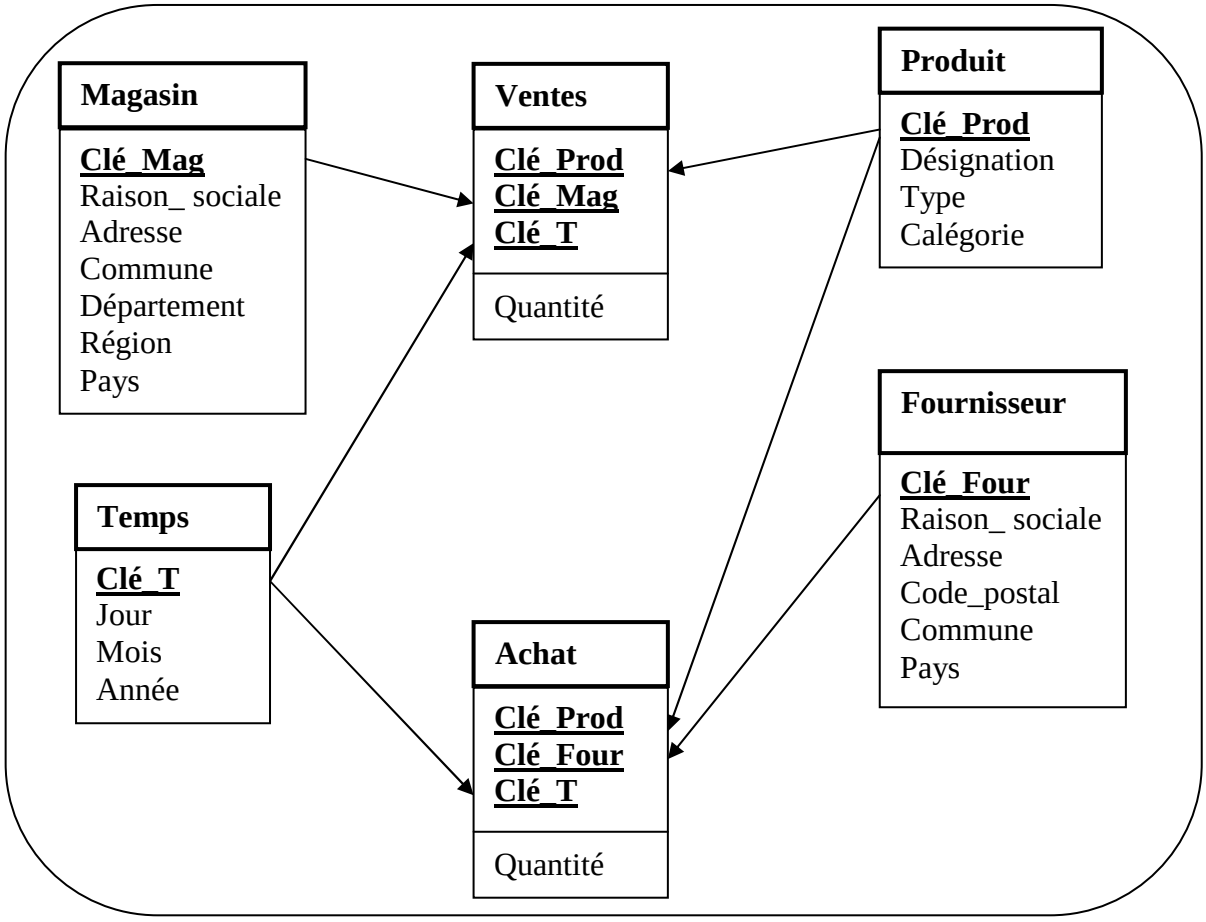


Figure 1.3 Exemple de modélisation en constellation

La relation de faits Ventes partage leurs dimensions Temps et Produits avec la table Achats. Néanmoins, la dimension Magasin appartient seulement à Ventes. Egalement, la dimension Fournisseur est liée seulement à la relation Achats.

2.4.2- Schéma multidimensionnel (Cube)

Le cube représente le concept central du modèle multidimensionnel, lequel est constitué des éléments appelés cellules qui peuvent contenir une ou plusieurs mesures. La localisation de la cellule est faite à travers les axes, qui correspondent chacun à une

dimension. La dimension est composée de membres qui représentent les différentes valeurs. En reprenant une partie du schéma en étoile de la figure 1.1, nous pouvons construire le schéma multidimensionnel suivant.

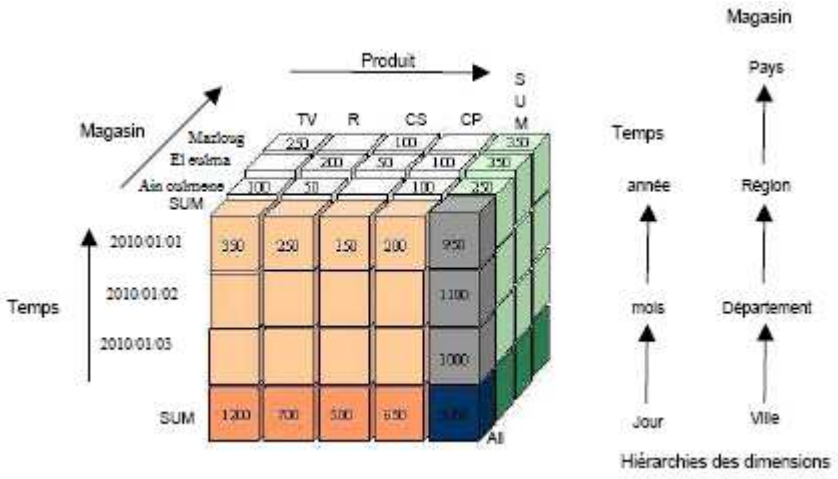


Figure 1.4 Exemple de schéma multidimensionnel [63]

La figure 1.4, présente un schéma multidimensionnel pour les ventes qui ont été réalisées dans les magasins pour les différents produits au cours d'un temps donné (jour). Par exemple, nous avons la quantité de 100 Téléviseurs vendus dans le magasin de Ain oulmene pendant le 1er janvier 2010.

2.4.3- Manipulation des données multidimensionnelles

La représentation des données multidimensionnelles sous forme d'une table de données permet de bien visualiser ces données, ce type de représentation est le plus courant [39]. Dans une table, nous représentons les différentes combinaisons des valeurs choisies pour constituer les noms de lignes et de colonnes. Néanmoins, quand le nombre de dimensions est supérieur à deux, l'utilisateur a des problèmes pour visualiser simultanément l'ensemble de l'information. Pour résoudre ce problème, nous devons disposer d'opérations pour manipuler les données et rendre possible la visualisation. Nous présentons les opérations pour la manipulation des données multidimensionnelles, en les divisant selon leur impact sur la façon de présenter les différentes vues des données analysées.

2.4.3.1- Opérations classiques

Ces opérations correspondent aux opérations relationnelles de manipulation des données :

La sélection : Résulte en un sous-ensemble de données qui respecte certaines conditions d'appartenance. Nous pouvons avoir une sélection avec des conditions soit sur les mesures, soit sur les membres. Par exemple, une sélection des ventes supérieures à 350 est une sélection sur les mesures, tandis qu'une sélection des ventes réalisées dans la région "Mezloug" de l'année "2010" est une sélection sur les membres d'une dimension.

La projection : Résulte en un sous-ensemble des attributs d'une relation, qui sont soit des dimensions, soit des niveaux de granularité. Dans les systèmes décisionnels, les opérations de sélection et de projection sont appelées souvent "slice-and-dice".

La jointure : Permet d'associer les données de relations différentes. Par exemple, en utilisant les tables 1.2 et 1.3, nous faisons une jointure sur la dimension Produit.

L'objectif est de représenter sur une même table la quantité de produits vendue et leur prix (Figure 1.5).

Ventes (01/01/2010)	Mag1	Mag2	Mag3
Téléviseur	**100**		**250**
Radio	**50**	**200**	
Caméscope		**50**	**100**
Camera photo	**100**	**100**	

Tableau 1.2 Ventes par magasin

Produit	Prix 01/01/2010
Téléviseur	**1**
Radio	**2**
Caméscope	**3**
Camera photo	**4**

Tableau 1 .3 prix des produits

Les opérations ensemblistes : D'union, d'intersection et de différence sont des opérations qui agissent sur des relations qui ont le même schéma.

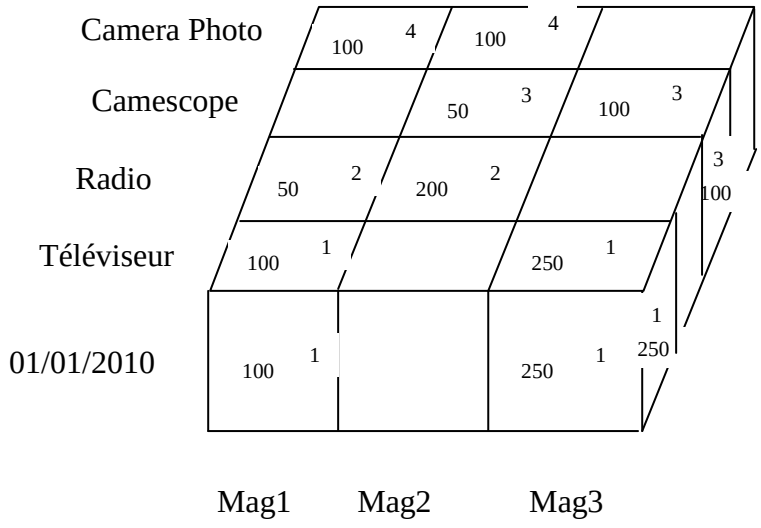


Figure 1.5 Exemple de l'opération de jointure [63]

2.4.3.2- Opérations agissant sur la structure

Les opérations agissant sur la structure permettent de présenter une vue (face du cube) différente en fonction de leur analyse, citons :

La rotation (rotate) : Consiste à pivoter ou à effectuer une rotation du cube, de manière à présenter une vue différente des données à analyser.

La permutation (switch) : Consiste à inverser des membres d'une dimension, de manière à permuter deux tranches du cube.

La division (split) : Consiste à présenter chaque tranche du cube en passant d'une représentation tridimensionnelle à une présentation tabulaire.

Nous remarquons que le nombre de tables résultantes de cette opération dépend du nombre de valeurs à l'intérieur de la dimension.

L'emboîtement (nest) : Permet d'imbriquer les membres d'une dimension. En utilisant cette opération, nous représentons dans une table bidimensionnelle toutes les données d'un cube quel que soit le nombre de dimensions.

L'enfoncement (push) : Consiste à combiner les membres d'une dimension aux mesures du cube et donc de représenter un membre comme une mesure.

L'opération inverse de retrait (pull) : Permet de changer le statut de certaines mesures, pour transformer une mesure en membre d'une dimension.

La factualisation (fold) : Consiste à transformer une dimension en mesure(s) ; cette opération permet de transformer en mesure l'ensemble des paramètres d'une dimension.

La paramétrisation (unfold) : Permet de transformer une mesure en paramètre dans une nouvelle dimension.

L'opération Cube : Permet de calculer des sous-totaux et un total final dans le cube (Figure 1.4).

2.4.3.3- Opérations agissant sur la granularité

Les opérations agissant sur la granularité des données analysées, permettent de hiérarchiser la navigation entre les différents niveaux de détail d'une dimension. Dans la suite nous traitons les deux opérations de ce type :

Le forage vers le haut (drill-up ou roll-up) : Permet de représenter les données du cube à un niveau plus haut de granularité en respectant la hiérarchie de la dimension. Nous utilisons une fonction d'agrégation (somme, moyenne,...), qui est paramétrée, pour indiquer la façon de calculer les données du niveau supérieur à partir de celles du niveau inférieur.

Le forage vers le bas (drill-down ou roll-down ou scale-down) : Consiste à représenter les données du cube à un niveau de granularité inférieur, donc sous une forme plus détaillée.

Ces types d'opérations ont besoin d'informations non représentées dans un cube, pour augmenter ou affiner des données, à partir d'une représentation initiale vers une représentation de granularité différente. Le forage vers le haut a besoin de connaître la fonction d'agrégation utilisée tandis que le forage vers le bas nécessite de connaître les données au niveau inférieur.

2.4.4- Serveurs OLAP (On-Line Analytical Processing)

Les systèmes décisionnels complets reposent sur la technologie OLAP, conçue pour répondre aux besoins d'analyse des applications de gestion. L'acronyme FASMI (*Fast Analysis of Shared Multidimensional Information*) permet de résumer la définition des produits OLAP. Cette définition fut utilisée pour la première fois en 1995 et depuis aucune autre définition n'est plus proche pour résumer le terme OLAP [43].

Fast : Le temps de réponse aux demandes des utilisateurs oscille entre 1 et 20 secondes. Les constructeurs utilisent des pré-calculs pour réduire les durées des requêtes.

Analysis : Le système doit pouvoir faire face à toutes les logiques d'affaires et de statistiques, ainsi que fournir la possibilité aux utilisateurs de construire leurs calculs et leurs analyses sans avoir à programmer. Pour cela, il y a des outils qui seront fournis par le constructeur.

Shared : Le système doit créer un contexte où la confidentialité est préservée et doit gérer les cas où plusieurs utilisateurs ont des droits en écritures. Ce point constitue la plus grosse faiblesse des produits actuels.

Multidimensional : C'est la caractéristique clé. Le système doit fournir des vues conceptuelles multidimensionnelles des données. Il doit supporter aussi les hiérarchies.

Informations : L'ensemble des données et les informations nécessaires pour un produit OLAP. Nous exposons dans la suite les divers types de stockage des informations dans les systèmes décisionnels.

2.4.4.1- ROLAP (Relational OLAP)

Dans les systèmes relationnels OLAP, l'entrepôt de données utilise une base de données relationnelle. Le stockage et la gestion de données sont relationnels. Toutefois, le modèle relationnel requiert des extensions pour supporter les requêtes d'analyses multidimensionnelles du niveau d'application. Le moteur ROLAP traduit dynamiquement le modèle logique de données multidimensionnel *M* en modèle de stockage relationnel *R* (la plupart des outils requièrent que la donnée soit structurée en utilisant un schéma en étoile ou un schéma en flocon de neige). Techniquement, le

moteur ROLAP exécute une transformation à partir d'une requête multidimensionnelle *m c*ontre *M* vers une requête relationnelle *r* contre *R*. L'efficacité du résultat de la requête est le facteur dominant pour la performance et le passage à l'échelle global du système [63]. Ainsi, les stratégies d'optimisation représentent le point principal qui distingue les systèmes ROLAP existants [11].

La technologie ROLAP a deux avantages principaux : (1) elle permet la définition de données complexes et multidimensionnelles en utilisant un modèle relativement simple, et (2) elle réduit le nombre de jointures à réaliser dans l'exécution d'une requête. Le désavantage est que le langage de requêtes tel qu'il existe, n'est pas assez puissant ou n'est pas assez flexible pour supporter de vraies capacités d'OLAP [54].

2.4.4.2- MOLAP (Multidimensional OLAP)

Les systèmes multidimensionnels OLAP utilisent une base de données multidimensionnelle pour stocker les données de l'entrepôt et les applications analytiques sont construites directement sur elle. Dans cette architecture, le système de base de données multidimensionnel sert tant au niveau de stockage qu'au niveau de gestion des données. Les données des sources sont conformes au modèle multidimensionnel, et dans toutes les dimensions, les différentes agrégations sont pré-calculées pour des raisons de performance [55].

Les systèmes MOLAP doivent gérer le problème de données clairsemées, quand seulement un nombre réduit de cellules d'un cube contiennent une valeur de mesure associée. La façon de résoudre cette problématique, par les produits commerciaux, représente un des plus importants critères pour les systèmes MOLAP. La plupart des systèmes existants utilisent l'approche du vecteur multidimensionnel pour le stockage de données, ainsi que la compression de l'espace de stockage pour les vecteurs de plus de trois dimensions. Les avantages des systèmes MOLAP sont basés sur les désavantages des systèmes ROLAP et elles représentent la raison de leur création. D'un côté, les requêtes MOLAP sont très puissantes et flexibles en termes du processus OLAP, tandis que, d'un autre côté, le modèle physique correspond plus étroitement au modèle multidimensionnel. Néanmoins, il existe des désavantages au

modèle physique MOLAP. Le plus important, à notre avis, c'est qu'il n'existe pas de standard du modèle physique.

2.4.4.3- HOLAP (Hybrid OLAP)

Un système HOLAP est un système qui supporte et intègre un stockage des données multidimensionnel et relationnel d'une manière équivalente pour profiter des caractéristiques de correspondance et des techniques d'optimisation.

Ci-dessous, nous traitons une liste des caractéristiques principales qu'un système HOLAP doit fournir [12] :

***La transparence du système* :** Pour la localisation et l'accès aux données, sans connaître si elles sont stockées dans un SGBD relationnel ou dimensionnel. Pour la transparence de la fragmentation,...

***Un modèle de données générales et un schéma multidimensionnel global* :**

Pour aboutir à la transparence du premier point, tant le modèle de données général que le langage de requête uniforme doit être fournis. Etant donné qu'il n'existe pas un modèle standard, cette condition est difficile à réaliser.

Une allocation optimale dans le système de stockage : Le système HOLAP doit bénéficier des stratégies d'allocation qui existent dans les systèmes distribués tels que : le profil de requêtes, le temps d'accès, l'équilibrage de chargement,...

Une re-allocation automatique : Toutes les caractéristiques traitées ci-dessus changent dans le temps. Ces changements peuvent provoquer la réorganisation de la distribution des données dans le système de stockage multidimensionnel et relationnel, pour assurer des performances optimales. Actuellement, la plupart des systèmes commerciaux utilisent une approche hybride. Cette approche permet de manipuler des informations de l'entrepôt de données avec un moteur ROLAP, tandis que pour la gestion des *data marts*, ils utilisent l'approche multidimensionnelle. Dans la figure 1.6, nous montrons une architecture en utilisant les types de serveurs ROLAP et MOLAP pour le stockage de données.

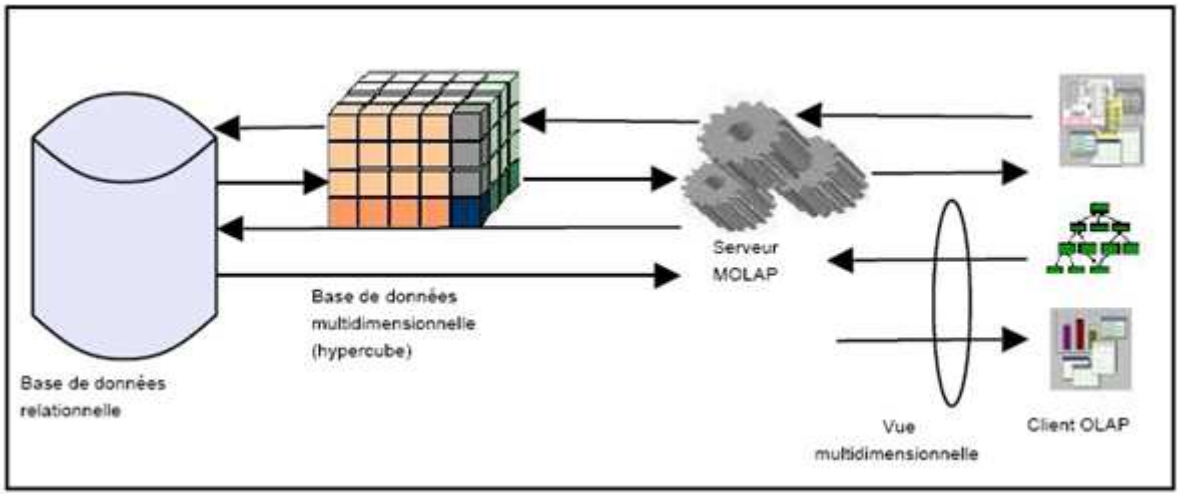


Figure 1.6 Architecture HOLAP [63]

3- Les Techniques d'optimisation des requêtes :

Ces techniques sont classées en trois catégories [70] :

3.1- Les index

Compte tenu de la complexité des requêtes décisionnelles et de la nécessité d'un temps de réponse court, plusieurs techniques d'indexation ont été développées pour accélérer l'exécution des requêtes.

3.1.1- Les techniques d'indexation

Les techniques d'indexation utilisées dans les bases de données de type (OLTP) ne sont pas bien adaptées aux environnements des entrepôts de données. En effet la plupart des transactions OLTP accèdent à un petit nombre de n-uplets, et les techniques utilisées (index B+ par exemple) sont adaptées à ce type de situation. Les requêtes décisionnelles adressées à un entrepôt de données accèdent au contraire à un très grand nombre de n-uplets (ce type de requête est encore appelé requêtes d'intervalle). Réutiliser les techniques des systèmes OLTP conduirait à des index avec un grand nombre de niveaux qui ne seraient donc pas très efficaces [59]. Un index peut être défini sur une seule colonne d'une relation, ou sur plusieurs colonnes d'une même relation. Nous appelons ce type d'index mono index. Il pourra être clustérisé ou non clustérisé. Nous pouvons également avoir des index définis sur deux relations comme les index de jointure [58] qui sont appelés multi-index.

Dans les entrepôts de données, les deux types d'index sont utilisés : index sur liste des valeurs, et index de projection (pour les mono index), index de jointure en étoile (star join index) pour les index multi-index.

a) Index sur liste des valeurs

Un index sur liste de valeurs est constitué de deux parties. La première partie est une structure d'arbre équilibré et la deuxième est un schéma de correspondance. Ce schéma est attaché aux feuilles de l'arbre et il pointe vers les n-uplets de la table à indexer. L'arbre est généralement de type B avec une variation de pourcentage d'utilisation. Deux types différents de schéma de correspondance sont utilisés. Le premier consiste en une liste de RowID associée à chaque valeur unique de la clé de recherche. Cette liste est partitionnée en blocs disque chaînés entre eux. Le deuxième schéma est de type bitmap. Il utilise un index binaire [59] représenté sous forme d'un vecteur de bits. Dans ce vecteur, chaque n-uplet d'une relation est associé à un bit qui prend la valeur 1 si le n-uplet est membre de la liste ou 0 dans le cas contraire. Un index binaire est une structure de taille réduite qui peut être gérée en mémoire, ce qui améliore les performances. De plus, il est possible d'exécuter des opérations logiques (par exemple les opérations ET, OU, XOR, NOT) de manière performante [59]. Cette technique d'indexation est appropriée lorsque le nombre de valeurs possibles d'un attribut est faible (par exemple l'attribut sexe qui peut prendre comme valeur masculin ou féminin) (voir figure 1.7).

Evidemment, le coût de maintenance peut être élevé car tous les index doivent être actualisés à chaque nouvelle insertion d'un n-uplet. L'espace de stockage augmente en présence de dimensions de grande cardinalité, parce qu'il faut gérer une quantité importante de vecteurs qui contiennent un grand nombre de bits avec la valeur 0. Pour éviter ce problème, des techniques de compression ont été proposées, comme le "run-length encoding".

Table Client

Nom	Age		Sexe
Ali	20		M
Nihad	25		F
Salim	18		M
Hichem	40		M
Rined	10		F

BM1
1
0
1
1
0

BM2
0
1
0
0
1

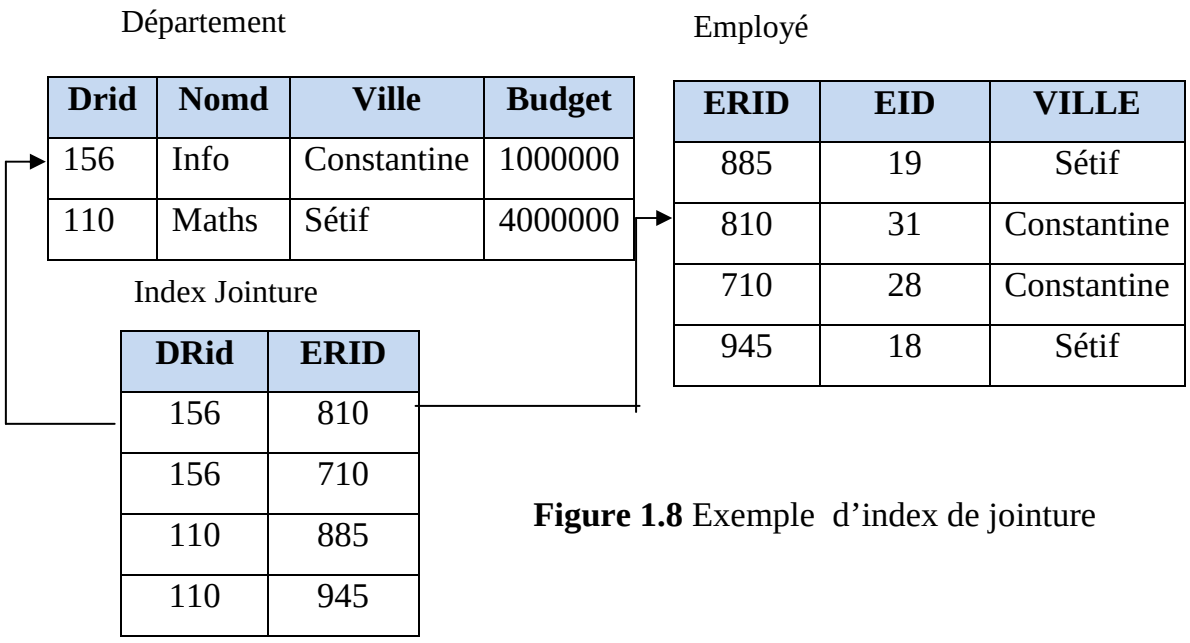
Figure 1.7 Index Binaire

Dans cette technique, une séquence de bits de la même valeur est représentée de manière compacte par une paire dont le premier élément est la valeur des bits et le deuxième le nombre de bits dans la séquence. L'utilisation de ce type de méthode dégrade les performances du système décisionnel à cause des traitements de compression et de décompression des index.

b) Index de jointure

Les requêtes complexes définies sur une base de données relationnelle demandent fréquemment des opérations de jointure entre plusieurs tables. L'opération de jointure est fondamentale dans les bases de données, et est très coûteuse en termes de temps de calcul lorsque les tables concernées sont grandes. Plusieurs méthodes ont été proposées pour accélérer ces opérations. Ces méthodes incluent les boucles imbriquées, le hachage, la fusion, etc. Valduriez a proposé des index spécialisés appelés index de jointure, pour pré-joindre des relations. Un index de jointure matérialise les liens entre deux relations par le biais d'une table à deux colonnes, contenant les RID (identifiant de n-uplet) des n-uplets joints deux par deux Figure 1.8).

Cet index peut être vu comme une jointure pré calculée. Créé à l'avance, il est implémenté par une relation d'arité 2. L'efficacité dépend du coefficient de sélectivité de jointure. Si la jointure a une forte sélectivité, l'index de jointure sera petit et aura une grande efficacité.



Drid	Nomd	Ville	Budget
156	Info	Constantine	1000000
110	Maths	Sétif	4000000

ERID	EID	VILLE
885	19	Sétif
810	31	Constantine
710	28	Constantine
945	18	Sétif

DRid	ERID
156	810
156	710
110	885
110	945

Figure 1.8 Exemple d'index de jointure

Ce genre d'index est souhaité pour les requêtes des systèmes OLTP car elles possèdent souvent des jointures entre deux tables [46]. (voir Par contre, pour les entrepôts de données modélisés par un schéma en étoile (schéma le plus couramment utilisé), ces index sont limités. En effet les requêtes décisionnelles définies sur un schéma en étoile possèdent plusieurs jointures (entre la table des faits et plusieurs tables de dimension). Il faut alors subdiviser la requête en fonction des jointures. Or le nombre de jointures possibles est de l'ordre de N!, N étant le nombre de tables à joindre (problème d'ordonnancement de jointure).

Pour résoudre ce problème, Red Brick [46] a proposé un nouvel index appelé index de jointure en étoile (star join index), adapté aux requêtes définies sur un schéma en étoile. Un index de jointure en étoile peut contenir toute combinaison de clés étrangères de la table des faits. Supposons par exemple que nous ayons un schéma en étoile modélisant les ventes au niveau d'un grand magasin.

Ce schéma contient une table des faits VENTES et trois tables de dimensions (TEMPS, PRODUIT et CLIENT). Un index de jointure en étoile peut être n'importe quelle combinaison contenant la clé de la table de faits et une ou plusieurs clés primaires des tables de dimensions. Ce type d'index est dit complet s'il est construit en joignant toutes les tables de dimensions avec la table des faits. Un index de jointure partiel est construit en joignant certaines des tables de dimensions avec la table des

faits. En conséquence, l'index complet est bénéfique pour n'importe quelle requête posée sur le schéma en étoile. Il exige cependant beaucoup d'espace pour son stockage. A ce stade de notre présentation, deux remarques s'imposent concernant l'index de jointure en étoile :

1. Comme son nom l'indique, ce type d'index est exclusivement adapté aux schémas en étoile [23]. Par contre, pour d'autres schémas comme le flocon de neige, ces index ne sont pas bien adaptés. Notons qu'il n'existe pas d'index de jointure efficace pour tous les schémas logiques de type ROLAP.
2. Dans la littérature, nous ne trouvons pas d'algorithme de sélection d'index de jointure en étoile pour un ensemble de requêtes. Ce problème est important car très souvent il n'est pas possible de construire un index de jointure en étoile complet. Il faut donc sélectionner un ou plusieurs index de jointure en étoile pour satisfaire au mieux les requêtes.

3.1.2- Le problème de sélection des index (PSI) :

Le PSI consiste, à partir d'un ensemble de requêtes décisionnelles et la contrainte d'une ressource donnée (l'espace, le temps de maintenance, etc.) à sélectionner un ensemble d'index afin de minimiser le coût d'exécution des requêtes. Ce problème a été reconnu par la communauté académique et industrielle. Les chercheurs ont modélisé le PSI comme un problème d'optimisation et ont proposé des heuristiques afin de trouver des solutions optimales ou quasi-optimales. Le groupe base de données de Microsoft a développé un outil pour sélectionner des index avec Microsoft SQL Server 7.0 [49]. Étant donné une charge constituée d'un ensemble de requêtes SQL, l'outil de sélection des index recommande d'une manière automatique un ensemble d'index pour cette charge. L'utilisateur peut spécifier des contraintes, par exemple une borne maximale pour l'espace alloué ou pour le nombre d'index. L'outil permet à l'utilisateur d'établir une analyse quantitative de l'impact de la recommandation proposée. Si l'utilisateur accepte les recommandations, l'outil crée (et/ou élimine) des index afin que les index recommandés soient matérialisés. L'architecture de l'outil de sélection des index proposé est illustrée dans la Figure 1.9.

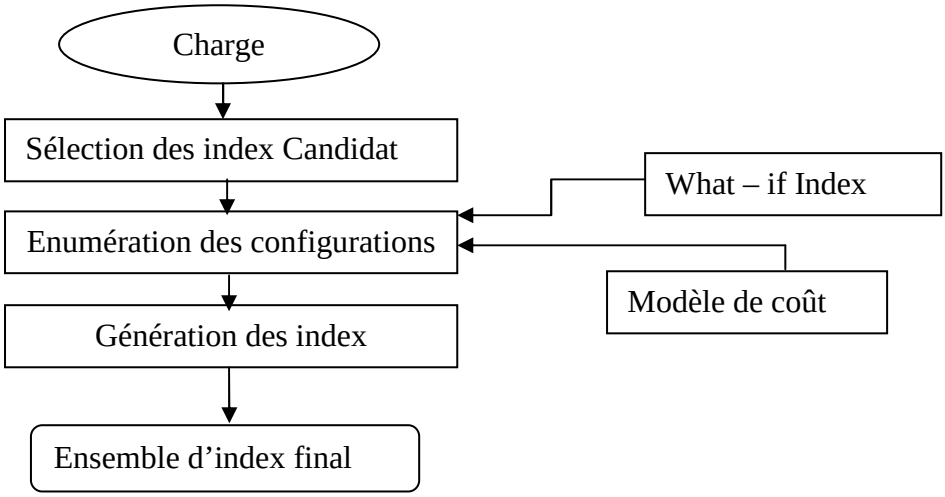


Figure 1.9 L'architecture de l'outil de sélection d'index

L'outil prend un ensemble de requêtes définies sur un schéma de base de données. Le traitement est itératif. Durant la première itération, il choisit les index sur une colonne (mono-index) ; dans la deuxième les index sur deux colonnes et ainsi de suite. L'algorithme de recherche d'index est testé en fonction de ces trois modules :

(1) La sélection des index candidats

(2) L'énumération des configurations .

(3) La génération des multi-index.

– Le module de sélection des index candidats permet de déterminer la meilleure configuration pour chaque requête d'une manière indépendante. Finalement, il fait l'union de ces configurations.

– Le module d'énumération des configurations : s'il existe n index candidats, et que l'outil doit sélectionner k parmi n index, le module d'énumération doit énumérer toutes les configurations, et à l'aide d'un modèle de coût sélectionner le meilleur ensemble de configurations garantissant un coût minimal.

Cet algorithme de sélection des index prend une requête à un moment donné et sélectionne tous les index possibles. Cependant, l'ensemble des index utilisant cette méthodologie pourra exiger beaucoup d'espace de stockage et des coûts de maintenance élevés.

Dans le but de minimiser les coûts de stockage et de maintenance, Chaudhuri et al. [50] ont proposé une technique appelée fusion d'index (index merging). Elle prend un ensemble d'index ayant une capacité d'espace S et fournit un nouvel ensemble d'index

ayant une capacité d'espace S′ inférieure à celle de départ (S′ < S). L'opération de fusion est guidée par un modèle de coût: la fusion est appliquée s'il y a une réduction dans le coût d'exécution des requêtes. La technique de fusion d'un ensemble d'index ressemble à la reconstruction des fragments verticaux d'une relation donnée. Le problème de fusion des index est formulé comme suit :

Entrées :

– Une configuration d'index initiale C = $\{I_1, I_2, ..., I_N\}$

– Un ensemble de requêtes Q = $\{Q_1, Q_2, ...,Q_p\}$

– Une borne maximale U pour le coût d'exécution des requêtes

Le but consiste à trouver une configuration minimale C′ =$\{J_1, J_2, ..., J_k\}$,k ≤ N tel que :

– coût (Q, C') ≤ U, (coût (Q, C') représente le coût(ou estimation) d'exécution de l'ensemble des requêtes de Q en présence de la configuration C′)

– C′ doit avoir un coût de stockage inférieur à celui de C.

3.2- La fragmentation

La fragmentation consiste en la division en plusieurs fragments (sous-ensembles de la relation) qui peuvent être non disjoints.

Deux sortes de fragmentation sont possibles : la fragmentation horizontale et la fragmentation verticale.

3.2.1- La fragmentation Verticale

La fragmentation verticale sert à diviser une relation en sous relations appelées fragments verticaux qui sont des projections appliquées à la relation. La fragmentation verticale favorise naturellement le traitement des requêtes de projection portant sur les attributs utilisés dans le processus de la fragmentation, en limitant le nombre de fragments à accéder. Son inconvénient est qu'elle requiert des jointures supplémentaires lorsqu'une requête accède à plusieurs fragments.

3.2.2- La fragmentation Horizontale

La fragmentation horizontale consiste à diviser une relation R en sous ensembles de n-uplets appelés fragments horizontaux, chacun étant défini par une opération de restriction appliquée à la relation. Les n-uplets de chaque fragment

horizontal satisfait une clause de prédicats. Le schéma de fragmentation de la relation R est donné par :

$H_1 = \sigma_{cl1}(R), H_2 = \sigma_{cl2}(R), ..., H_q = \sigma_{clq}(R)$, ou cl_i est une clause de prédicats.

La reconstruction de la relation R à partir de ces fragments horizontaux est obtenue par l'opération d'union de ces fragments. La fragmentation horizontale se décline en deux versions [3] : les fragmentations primaire et dérivée.

a) *La fragmentation primaire* d'une relation est effectuée grâce à des prédicats de sélection définis sur la relation [37]. La fragmentation horizontale dérivée s'effectue avec des prédicats de sélection définis sur une autre relation.

b) *La fragmentation horizontale dérivée :* Une relation S (relation membre) peut contenir une clé étrangère vers une relation R (relation propriétaire). La fragmentation horizontale dérivée est définie sur la relation membre S en fonction des fragments horizontaux de la relation propriétaire R.

   Plus formellement, soit R une relation propriétaire horizontalement fragmentée en m fragments horizontaux $\{R_1, R_2, ..., R_m\}$. Soit S une relation membre. Les fragments horizontaux dérivés $\{S_1, S_2, ..., S_m\}$ de S sont définis par l'opération de semi-jointure suivante : $S_i = S \ltimes R_i$. Trois informations sont nécessaires pour effectuer une fragmentation horizontale dérivée :

   (1) Le nom de la relation membre et de la relation propriétaire.

   (2) Le nombre de fragments horizontaux de la relation membre.

   (3) La qualification de la jointure entre ces deux relations.

La fragmentation horizontale primaire favorise le traitement des requêtes de restriction portant sur les attributs utilisés dans le processus de la fragmentation. La fragmentation horizontale dérivée est utile pour le traitement des requêtes de jointure.

3.3- Les vues matérialisées

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 pré calculant 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'accès aux vues matérialisées et sont ainsi exécutées plus rapidement. Les vues dans le contexte OLTP ont été largement utilisées pour répondre à plusieurs rôles : la sécurité, la confidentialité, l'intégrité référentielle, etc.

Les vues matérialisées peuvent être utilisées pour satisfaire plusieurs objectifs, comme l'amélioration de la performance des requêtes ou la fourniture des données dupliquées. Deux problèmes majeurs sont liés aux vues matérialisées : (1) le problème de maintenance des vues matérialisées et (2) le problème de sélection des vues. Nous abordons ces deux problèmes dans les sections suivantes.

3.3.1- Maintenance de vues matérialisées

Un entrepôt de données contient un ensemble de 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 obsolètes 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 desquels il a été dérivé ont été supprimés ou modifiés. Afin de résoudre ce problème d'inconsistance des données, une procédure de maintenance des vues doit être mise en place.

Trois stratégies fondamentales ont été proposées. Dans la première stratégie, les vues sont mises à jour périodiquement [2]. Dans ce cas, ces vues peuvent être considérées comme des photographies (snapshots). Dans la deuxième stratégie [31], les vues sont mises à jour immédiatement à la fin de chaque transaction. Dans la dernière stratégie, les modifications sont propagées d'une manière différée. Dans ce cas, une vue est mise à jour uniquement au moment où elle est utilisée par une requête d'un utilisateur.

Quelle que soit la stratégie adoptée, la maintenance pourrait consister à simplement recalculer le contenu des vues matérialisées à partir des tables sources. Cependant, cette approche est complétement inefficace (très coûteuse). En effet, une bonne maintenance des vues est réalisée lorsque les changements (insertions, suppressions, modifications) effectués dans les tables sources peuvent être propagés aux vues sans obligation de recalculer complètement leur contenu.

Plusieurs techniques ont été proposées dans la littérature pour répondre à ce besoin : la maintenance incrémentale, la maintenance autonome des vues, et la maintenance des vues en batch.

3.3.2- La sélection des vues matérialisées

Dans l'environnement d'un entrepôt de données, il est généralement possible d'isoler un ensemble de requêtes à privilégier. L'ensemble des vues matérialisées doit être déterminé en fonction de cet ensemble de requêtes.

Le problème de sélection des vues matérialisées (PSV) peut être vu sous deux angles en fonction du type de modèles de données : (1) le PSV de type ROLAP et (2) le PSV de type MOLAP.

– Dans le PSV de type MOLAP, nous considérons le cube de données comme la structure primordiale pour sélectionner les vues matérialisées. Chaque cellule du cube est considérée comme une vue potentielle. Notons que le cube de données est un cas spécial d'entrepôt, ne contenant que les requêtes ayant des agrégations sur les relations de base.

– Dans le PSV de type ROLAP, chaque requête est représentée par un arbre algébrique. Chaque noeud (non feuille) est considéré comme une vue potentielle. Ce type de PSV est plus général que le premier type.

Il existe trois possibilités pour sélectionner un ensemble de vues [32] :

1. matérialiser toutes les vues : Dans le cube de données cette approche consiste à matérialiser la totalité du cube, tandis que dans le cas du ROLAP, elle consiste à matérialiser tous les noeuds intermédiaires des arbres algébriques représentant les requêtes. Cette approche donne le meilleur temps de réponse pour toutes les requêtes. Mais stocker et maintenir toutes les cellules/noeuds intermédiaires est impraticable pour un entrepôt important. De plus, l'espace utilisé par les vues peut influencer la sélection des index.

2. ne matérialiser aucune vue : Dans ce cas nous sommes obligés d'accéder aux données des relations de base. Cette solution ne fournit aucun avantage pour les performances des requêtes.

3. matérialiser seulement une partie du cube/des noeuds : Dans un cube, il existe une certaine dépendance entre les cellules, c'est à dire que la valeur de certaines cellules peut être calculée à partir des valeurs d'autres cellules. Dans le cas d'un système ROLAP, on trouve également cette dépendance dans les arbres algébriques. Il est alors souhaitable de matérialiser les parties partagées (cellules ou noeuds) par plusieurs requêtes. Cette approche a pour but de sélectionner les cellules ou les noeuds partagés. Cette solution semble la plus intéressante par rapport aux deux approches précédentes.

Quel que soit le type de PSV, ce dernier peut être défini de la manière suivante [21, 32] : étant donné une contrainte de ressource S (capacité de stockage, par exemple), le PSV consiste à sélectionner un ensemble de vues $\{V_1, V_2, ..., V_k\}$ minimisant une fonction objectif (coût total d'évaluation des requêtes et/ou coût de maintenance des vues sélectionnées) et satisfaisant la contrainte.

Dans notre étude on est intéressé par la résolution de ce problème (PSV).

4- Conclusion

Ce chapitre présente les concepts de base de l'entrepôt de données (Data Warehouse) et la différence entre le processus transactionnel en ligne (OLTP) qui gèrent les bases de données représentant le système opérationnel de l'entreprise et le processus d'analyse en ligne de données qui exploite l'entrepôt de données pour les applications d'aide à la décision.

Aussi, ce chapitre décrit la modélisation multidimensionnelle qui nous permettra de faire la conception d'un entrepôt de données, ainsi que le langage de manipulation de données de l'entrepôt. Comme les entrepôts de données sont des bases de données larges, ceci rend la manipulation des requêtes décisionnelles (contenant des agrégations) lourde, à cet effet des techniques d'optimisations ont été développé pour dépasser cet inconvénient.

Dans notre étude on s'est concentré sur la technique des vues matérialisées, Et spécialement la sélection des vues à matérialiser. On va décrire dans le chapitre suivant en détail les contraintes exigées lors de la sélection des vues à matérialiser.

1- Introduction

Les requêtes OLAP habituellement nécessitent l'agrégation (GROUP BY). Puisque les résultats d'une certaine requête OLAP sont si chers dans son calcul, l'agrégation des vues de données est souvent pré calculée et enregistrée pour accélérer le traitement de requêtes dans le futur. Dans le meilleur des cas toutes les vues seraient pré calculées et rendu disponible pour répondre à des requêtes globales, mais réellement il y a des contraintes exigées auparavant sur lesquelles des vues peuvent être matérialisées tel que la taille de l'ensemble de vues matérialisées ou le temps nécessaire pour appliquer les mises à jour à ces vues.

Le problème de sélection d'un ensemble de vues pour la matérialisation est connu comme *problème de sélection de vues*. Dans ce problème, on souhaite sélectionner un ensemble de vues pour la matérialisation qui minimise un ou plusieurs objectifs, probablement sur le sujet d'un ou plusieurs contraintes. Beaucoup de variantes du problème de sélection de vue ont été étudié.

Ce chapitre est consacré à la représentation du problème de sélection des vues et l'état de l'art des solutions développées pour ce problème. Donc, en premier lieu, nous décrirons les contraintes exigées dans le processus de sélection des vues à matérialiser.

Ensuite les outils mathématiques utilisés dans le développement des solutions, enfin nous étudierons les différents algorithmes proposés.

2- Les fonctions de coût et les contraintes du PSV

Pour sélectionner des vues matérialiser dans un entrepôt de donnes, on essaye de satisfaire un ou plusieurs buts de conception, et qui sont subdivisé en deux selon [69].

Le premier but est la minimisation d'une fonction de coût. Le second est de satisfaire ou de respecter les contraintes exigées soit par le système ou l'utilisateur.

2.1- Les fonctions de coût

La plupart des approches comportent dans leurs buts de conception la minimisation d'une fonction de coût. Ce qui suit sont des exemples de fonctions de coût à minimiser :

2.1.1- Coût d'évaluation de requête

Les entrées de PSV (problème de sélection de vues) sont souvent des requêtes qui doit être satisfaire par l'entrepôt de données. L'évaluation du coût de l'ensemble de requête est égale à la somme d'évaluation du coût de chaque entrée (requête) réécrite (partiellement ou complètement) au-dessus des vues matérialisées. Cette somme peut également être pondérer, chaque poids indique la fréquence ou l'importance de la requête correspondante. Plusieurs approches visent à minimiser l'évaluation de coût de requête [25].

2.1.2- Coût de maintenance des vues.

Le coût de maintenance des vues égal à la somme du coût de propagation de chaque changement de relation source en vues matérialisées. Cette somme peut être pondérer, chaque pondération indique la fréquence de propagation des changements de la relation source correspondante. Les expressions de maintenance peuvent être évaluées plus efficacement si elles peuvent être réécrites partiellement à travers des vues qui sont déjà matérialisé dans l'entrepôt de données. De plus, l'accès aux sources de données distantes et les transmissions de données coûteuses sont réduits. [48] a proposé d'ajouter des vues auxiliaires pour réduire le coût de maintenance des vues. Ces vues peuvent être matérialisées de manière permanente ou transitoire. Les vues matérialisées transitoires sont employées seulement pendant le processus de maintenance et elles sont détruites après [42]. Évidemment, le maintien des vues auxiliaires engendre un coût de maintenance additionnel. Ross [47] dérivent des vues auxiliaires pour matérialiser en permanence afin de réduire au minimum le coût de maintenance des vues.

2.1.3- Coût opérationnel.

La réduction au minimum du coût d'évaluation de requêtes et de maintenance des vues sont des exigences en conflits. Un coût de maintenance bas peut être obtenu par la réplication des relations de source dans l'entrepôt de données. Dans ce cas, le coût d'évaluation des requêtes est élevé puisque les requêtes doivent être calculées à partir des relations sources répliquées. Un coût d'évaluation de requête bas peut être obtenu en matérialisant toutes les requêtes d'entrée dans l'entrepôt de données. Dans ce cas, toutes les requêtes d'entrée peuvent être répondues par une simple consultation mais le coût de maintenance des vues deviendra élevé. Les requêtes d'entrée peuvent se chevauchées, c.-à-

d., elles peuvent partager beaucoup de sous expressions communes. Par la matérialisation de sous expressions communes et d'autres vues au-dessus des relations sources, il est possible, en général, de réduire le coût de maintenance des vues. Ces réductions doivent être équilibrées contre une évaluation plus élevée de coût de requête. Pour cette raison, on peut choisir de minimiser une combinaison linéaire de coût l'évaluation de requête et de maintenance de vue qui s'appelle le ***coût opérationnel***. La plupart des approches ont essayé de réduire au minimum le coût opérationnel [20].

2.1.4- Taille totale de vue.

Dans un entrepôt de données distribué où les vues matérialisées sont stockées à distance, le temps de transmission de données à travers le réseau habituellement représente un goulot d'étranglement est. Dans ce cas, le concepteur est intéressé de réduire au minimum la taille de l'ensemble des vues matérialisées qui répondent à toutes les requêtes d'entrée [5].

2.2- Les contraintes

Les contraintes sont classifiées en deux, l'une se concentre sur le système et l'autre sur l'utilisateur [69] :

2.2.1- Les Contraintes orientées Système

Des contraintes orientées système sont dictées par les restrictions du système, et qui sont transparentes aux utilisateurs.

- **La contrainte d'espace.**

Bien que la dégradation du coût d'espace disque permet un stockage massif des données, on ne peut pas considérer que l'espace disque est illimité. La contrainte de l'espace exige que la taille des vues matérialisées sélectionnées ne dépasse pas l'espace réservé à eux dans l'entrepôt de données. Des contraintes de l'espace sont adoptées dans beaucoup de travaux [24].

- **La contrainte de coût de maintenance de vue.**

Dans beaucoup de cas pratiques le facteur s'abstient dans la matérialisation de toutes les vues dans l'entrepôt de données n'est pas la contrainte de l'espace mais le coût de maintenance des vues. Habituellement, les entrepôts de données sont mis à jour périodiquement, par exemple dans la nuit, dans une grande transaction de mise à jour en

lots. Par conséquent, la fenêtre de mise à jour doit être suffisamment courte pour que l'entrepôt de données soit disponible pour interrogation et analyse pendant la journée. La contrainte de coût de maintenance des vues déclare que tout le coût de maintenance des vues devrait être moins qu'une quantité donnée du temps de maintenance des vues. [19] ont pris en considération la contrainte de coût de maintenance dans la sélection des vues matérialisées.

- **Auto maintenabilité.**

Une vue matérialisée est auto maintenable si elle peut être maintenu, pour n'importe quels exemple de relations de source dont il est définie, et pour tous les changements de relation source, en utilisant seulement ces changements, la définition de vue, et la matérialisation de vue. La notion est prolongée à un ensemble de vues d'une façon directe. En ajoutant des vues auxiliaires à un ensemble de vues matérialisées, on peut rendre tout l'ensemble de vue auto maintenable. Il y a différentes raisons pour rendre des vues auto-maintenable:

(a) Des relations de source à distance n'ont pas besoin d'être entrées en contact alternativement pour l'évaluation des expressions de maintenance pendant la mise à jour des vues.

(b) Les anomalies dues aux changements concourants sont éliminées et le processus de maintenance des vues est simplifié.

(c) Les vues matérialisées peuvent être maintenue efficacement même si les sources ne peuvent pas répondre à des requêtes (par exemple systèmes de legs), ou si elles sont temporairement indisponibles (par exemple dans les systèmes mobiles). En ajoutant des vues auxiliaires à un ensemble de vues matérialisées, l'ensemble entier des vues peut être rendu auto maintenable. L'auto maintenance peut être trivialement réalisée en repliant dans l'entrepôt de données toutes les relations source utilisées dans les définitions des vues. L'auto maintenance est vue comme une contrainte exige que l'ensemble des vues matérialisées adoptées soit auto-maintenable.

Répondre sur les entrées des requêtes en utilisant exclusivement les vues matérialisées

Cette contrainte exige l'existence d'une réécriture complète de l'entrée de la requête (dont sont initialement définies au-dessus des relations de source) au-dessus des vues matérialisées. Clairement, si cette contrainte est satisfaite, les sources de données distantes n'ont pas besoin d'être entrées en contact pour évaluer des requêtes. De cette manière, la

transmission de données coûteuse de l'entrepôt de données aux sources et réciproquement sont évitées. Quelques approches adoptent un environnement centralisé d'entrepôt de données où les relations source sont présentes dans le site de l'entrepôt. Dans ce cas, la réponse aux requêtes en utilisant les vues matérialisées est trivialement garantie par la présence des relations sources. Cette approche garantie en même temps l'auto-maintenance des vues matérialisées.

2.2.2- Les contraintes orientées utilisateur

Les contraintes orientées utilisateur expriment les exigences de l'utilisateur [69].

- **La Contrainte d'exactitude de données de réponse.**

La contrainte d'exactitude de données de réponse fixe une borne supérieure pour le temps écoulé entre le point du temps de la réponse à une requête retournée à l'utilisateur et le point du temps des plus récents changements des relations source qui sont pris en considération dans le calcul et qui sont lus (ce temps reflète l'actualité des données de réponse). Les Contraintes d'exactitude sont associées à chaque relation source de chaque définition d'entrée de requête. La borne supérieure dans une contrainte d'exactitude de données de réponse (exactitude minimale exigée) est placée par les utilisateurs selon leurs besoins. Cette formalisation des contraintes d'exactitude de données permet de déclarer les contraintes d'exactitude au niveau des requêtes et pas au niveau de la vue matérialisée de même cas dans quelques approches. Par conséquent, des contraintes d'exactitude peuvent être exploitées par les algorithmes de sélection de vues dans l'entrepôt de données où les requêtes sont des entrées, alors que les vues matérialisées sont des sorties (et ne sont pas donc disponibles). En outre, elle laisse indiquer différentes contraintes d'exactitude pour les différentes relations dans la même requête.

- **La contrainte du temps de réponse à la requête**

Cette contrainte définit que le temps nécessaire pour évaluer une requête d'entrée en utilisant les vues matérialisées dans le DW ne devrait pas dépasser une limite donnée. La limite pour chaque requête est donnée par les utilisateurs et reflète leur besoin pour une réponse rapide. Pour quelques requêtes des réponses rapides peuvent être exigées, tandis que pour d'autres le temps de réponse peut ne pas être prédominant.

2.3– Les modèles de coût de PSV :

On peut décrire le problème de sélection de vue comme suit : Comment sélectionner un ensemble approprié des vues matérialisées à partir d'un graphe de vues G, de sorte que le total du coût de traitement des requêtes supportées et le total du coût de maintenance de ces vues matérialisées soient minimaux. Soit un graphe G, et M un ensemble de vues à matérialiser dans G, et f_q la fréquence d'exécution de requêtes, et f_s la fréquence de mise à jour des relations de bases, respectivement.

De plus, pour chaque $v \in M$, soit E ($Gq_i(v)$) et M($Gs_i(v)$) le coût d'accès pour la requête Q en utilisant v et son coût de maintenance qui se base sur les changements de la relation de base S respectivement (c'est à dire, $v \in QN$ est l'ensemble de requêtes et $s \in SRN$ est l'ensemble de relations de bases).Alors le coût de traitement de requête sera [62]:

$$E\big(G(v)\big) = \sum_{i=1}^{M} fQ_i E(GQ_i(v)) \quad \text{.....(1)}$$

Et le coût d'entretien des vues matérialisées soit :

$$M(G(v)) = \sum_{i=1}^{n} fS_i M(GS_i(v)) \quad \text{.....(2)}$$

Ainsi le coût global pour matérialiser une vue v est :

$$\text{totalcoût}(v) = E(G(v)) + M(G(v)) \quad \text{..... (3)}$$

Ainsi, le coût total de matérialisation d'un ensemble de vues M est égale à:

$$\text{coût total} = \sum_{v \in M} \text{totalcoût}(v) \quad \text{.....(4)}$$

Les équations (1) et (2) peuvent être utilisées pour l'optimisation multi objectives et l'équation (3) peut être utilisée pour l'optimisation d'un unique objectif.

3- Fondements mathématique

La première étape dans la résolution du problème de sélection de vues est la formulation mathématique. Dans la formulation mathématique du problème de sélection de vues, des structures de données sont exigées pour représenter la sélection de vues. Pour ceci le concept de mathématique discret et la théorie de graphe et même la théorie des ensembles, et les treillis sont généralement employés. Dans les sous sections suivantes les principes fondamentaux des concepts mentionnés ci-dessus sont discutés avec l'aide d'un exemple.

3.1- Algèbre relationnel

L'algèbre relationnel descend de la catégorie du langage d'interrogation procédurale. Les requêtes dans l'algèbre relationnel se composent en utilisant une collection d'opérateurs. La propriété fondamentale dans l'algèbre relationnel est que chaque opérateur accepte (un ou deux) instances de relation comme arguments et retourne une instance de relation comme résultat. Cette propriété facilite la composition des opérateurs pour former une requête complexe. Une expression d'algèbre relationnel est périodiquement définie pour être une relation, soit un opérateur unaire appliqué à une expression simple, ou un opérateur binaire appliqué à deux expressions.

Certains opérateurs de base qui sont impliqués dans le traitement de la requête et leur occurrence jouent un rôle important dans le processus d'optimisation de requête, et dans le plan d'évaluation de requête. Les opérateurs fondamentaux de l'algèbre relationnel sont Sélection (σ), Projection (π), Union ($\cup$), Intersection ($\cap$),Produit Cartésien ($\times$),la différence (-), et jointure (∞).

3.2- Graphe orienté acyclique dirigé (DAG)

En théorie des graphes, un graphe orienté acyclique (en anglais directed acyclic graph ou DAG) est un graphe orienté qui ne possède pas de cycle. Il exprime formellement un outil traditionnel d'analyse.

Les DAG apparaissent dans les modèles où il ne contient pas un nœud qui a un chemin à lui-même

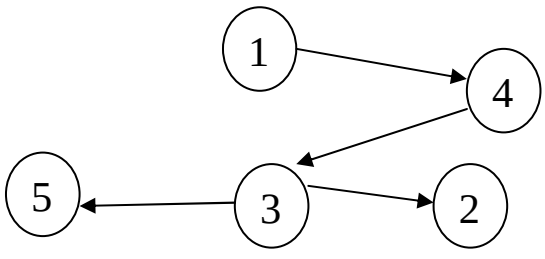


Figure 2.1 Graphe orienté acyclique

Les Graphes acycliques dirigés peuvent être employés pour représenter un certain nombre de relations intéressantes. Ceci inclut des arbres, mais il est moins général que la classe de tous les graphes dirigés.

3.3- Graphe AND/OR

Beaucoup de problèmes complexes peuvent être décomposés en série de sous problèmes tels que la solution de tous ou de certains résultant dans la solution du problème original. Ces sous problèmes peuvent être décomposés plus en sous problèmes et ainsi de suite jusqu'à ce que les problèmes restants soient suffisamment primitifs quant à être trivialement soluble.

Cette décomposition du problème complexe en plusieurs sous problèmes peut être représentée par un graphe orienté comme structure en laquelle les nœuds représentent les problèmes et les descendants d'un nœud représentent les sous problèmes liés à lui.

Le graphe AND est représenté dans la fig.2.2, un problème A qui peut être résolu en résolvant les deux sous problèmes B et C ou un seul sous problème D ou E.

Les groupes des sous problèmes qui doivent être résolus afin d'impliquer une solution au nœud parent sont joints ensemble par un arc à travers le perspectif bord c.-à-d. L'arc à travers les bords < A, B > et < A, C >.

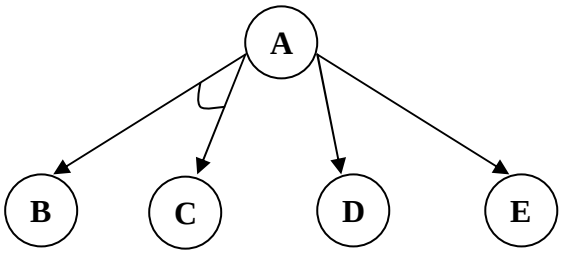


Figure2.2 Graphe AND

En introduisant les nœuds factices comme dans la figure 2.3, tous les nœuds peuvent être construits d'une manière que leurs solutions exigent que tous les descendants doivent être résolus ou seulement un descendant doit être résolu.

Les nœuds du premier type sont appelés les nœuds AND et l'autre type s'appelle les nœuds OR. Les nœuds À et de fig.2.3 sont des nœuds OR tandis que le nœud A est un nœud AND. Les nœuds AND seront dessinés avec un arc à travers tous les bords partant du nœud. Des nœuds qui n'ont pas des descendants se nomment borne [62].

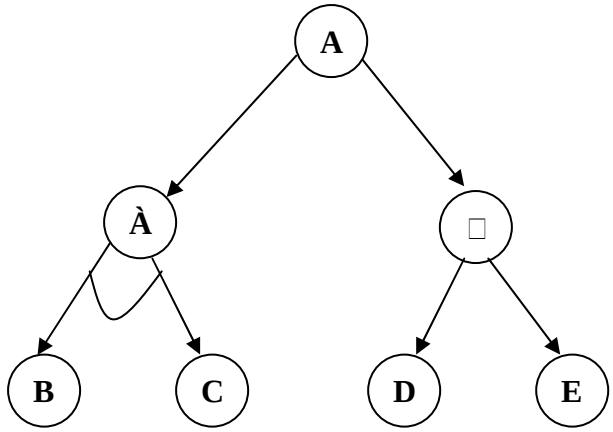


Figure 2.3 les Graphes représentant le problème AND/OR

Définition 3

Un graphe d'une requête (ou vue) de type AND est celui dans lequel la requête possède un plan d'exécution unique.

Définition 4

Un graphe d'une requête (ou vue) de type OR est celui dans lequel la requête possède plusieurs plans d'exécution.

3.4- Treillis

Un cube de données permet à des données d'être modelées et affichées dans des dimensions multiples. Il est défini par des dimensions et des faits. La propriété intéressante des cubes est que les données de n-D peuvent être représentées comme série de (n-1)-D cubes. Les applications de support de décision entraînent des requêtes complexes sur des bases de données larges. Puisque le temps de réponse devrait être petit, l'optimisation de requête deviendra critique. Les utilisateurs typiquement voient les données en tant que cubes de données multidimensionnels. L'opérateur C impose un ordonnancement partiel aux requêtes. Ceci est lié aux vues d'un problème de cube de données qui forment un treillis. Afin d'être un treillis, deux éléments quelconques (des vues ou des requêtes) doivent avoir au moins la limite supérieure et une plus grande limite inférieure selon l'ordonnancement C. Cependant, dans la pratique, on a besoin seulement de la prétention que C est un ordre partiel et qu'il y a un élément supérieur : une vue sur laquelle chaque vue est dépendante à elle. Considérant deux requêtes Q_1 et Q_2, $Q_1 \preceq Q_2$ peut être défini si Q_1 peut être répondu en utilisant seulement les résultats de Q_2.Alors on dit que Q_1 dépend de Q_2.Dans la plupart des applications, les dimensions d'un cube de données se composent plus d'un attribut et les

dimensions sont organisées comme hiérarchies de ces attributs. Un simple exemple est montré dans la figure 2.4, il organise la dimension temps dans la hiérarchie :jour, mois et année. En présence des hiérarchies, le treillis de dépendance est plus complexe qu'un treillis de hyper cube. Les hiérarchies introduites des dépendances de requête qu'on doit expliquer pour déterminer quelles requêtes à matérialiser. Quelques définitions s'imposent.

Définition 1

Une relation de dépendance entre les requêtes : Soient Q_i et Q_j deux requêtes (vues). On dit que $Q_i \preceq Q_j$ si et seulement si Q_i peut être évaluée en utilisant seulement les résultats de la requête Q_j.

Définition 2

Un treillis est construit de la façon suivante : les nœuds de ce treillis représentent les vues (agrégées sur certaines dimensions) et un arc existe entre deux nœuds V_i et V_j si elles sont dépendantes ($V_i \preceq V_j$). Le nœud V_i est un nœud ancêtre et V_j un descendant. Les ancêtres et les descendants d'un nœud Vi du treillis sont définis comme suit [62]:

$$\text{Ancêtre}(V_i) = \{V_j | V_i \preceq V_j\}$$

$$\text{Descendant}(V_i) = \{V_j | V_j \preceq V_i\}.$$

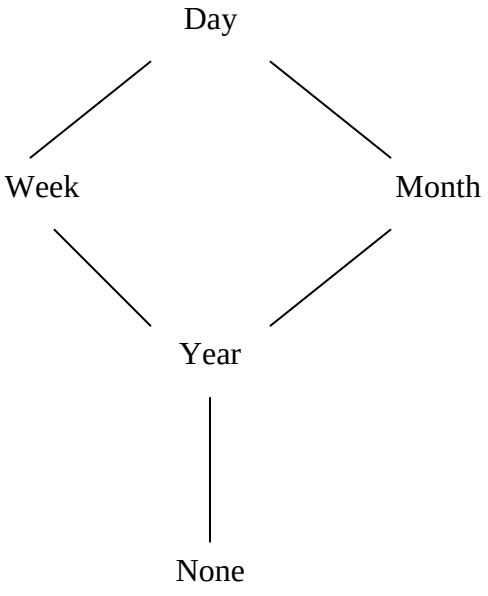


Figure 2.4 Exemple de Treillis

4- Les algorithmes de sélection des vues

La plupart des approches sur les problèmes de sélection de vues évitent des algorithmes exhaustifs. Les algorithmes adoptés tombent dans deux catégories [70]: déterministe et aléatoire.

- ✓ Dans la première catégorie appartiennent les algorithmes gloutons avec la garantie de performance [25], 0-1 algorithmes de programmation de nombre entier [61], A * algorithmes [21], et divers autres algorithmes heuristiques [48].
- ✓ Dans la deuxième catégorie appartiennent les algorithmes de recuit simulés [33], algorithmes itératifs d'amélioration [33], et algorithmes génétiques [36].

Les deux catégories d'algorithmes exploitent les particularités du problème de sélection de vues et les spécifiques et les restrictions de la classe des requêtes considérées. Les algorithmes proposés pour la sélection des vues peuvent également être classés en trois catégories, en fonction du type de contrainte qu'ils utilisent : (1) algorithmes sans aucune contrainte [26, 34, 2, 30], (2) algorithmes dirigés par la contrainte d'espace [14] et (3) algorithmes dirigés par la contrainte du temps total de maintenance des vues [14].

4.1- Les algorithmes sans aucune contrainte

Il existe plusieurs approches pour la sélection des vues sans contraintes. Nous en retenons deux, qui sont l'approche de Yang et al. [64] proposée dans le contexte ROLAP, et l'approche de Baralis [13] proposée dans le contexte MOLAP.

Les travaux de Yang et al. [64] Les auteurs ont développé un algorithme de sélection des vues dans un contexte ROLAP statique. Les auteurs partent du principe suivant : la principale caractéristique des requêtes décisionnelles est qu'elles utilisent souvent les résultats de certaines requêtes pour répondre à d'autres requêtes [13]. On peut tirer de cette caractéristique que les requêtes décisionnelles partagent certaines expressions. L'algorithme de Yang et al. Procède de la façon suivante :

Chaque requête est représentée par un arbre algébrique. Etant donné que chaque requête peut avoir plusieurs arbres algébriques, les auteurs sélectionnent l'arbre optimal (en fonction d'un modèle de coût). Une fois les arbres optimaux identifiés, l'algorithme essaye de trouver des expressions communes entre ces arbres (ou nœud partagé). Finalement, les arbres sont fusionnés en un seul graphe, appelé plan multiple d'exécution des vues en utilisant les nœuds partagés identifiés.

Ce graphe a plusieurs niveaux. Les feuilles sont les tables de base de l'entrepôt et représentent le niveau 0. Dans le niveau 1, nous trouvons des nœuds représentants les résultats des opérations algébriques de sélection et de projection. Dans le niveau 2, les nœuds représentent les opérations ensemblistes comme la jointure, l'union, etc. Le dernier niveau représente les résultats de chaque requête. Chaque nœud intermédiaire de ce graphe est étiqueté par le coût de l'opération algébrique (sélection, jointure, union, etc.) et le coût de maintenance. Ce graphe est utilisé pour rechercher l'ensemble des vues dont la matérialisation minimise la somme des coûts d'évaluations des requêtes et de maintenance des vues. La solution prend en considération l'existence de plusieurs expressions possibles pour une requête. Chaque nœud intermédiaire est considéré comme une vue potentielle.

Exemple1 : Soit un schéma d'un entrepôt ayant cinq tables et sur lequel trois requêtes sont définies.

La Figure 2.5 montre que les requêtes Q1 et Q2 ont une expression commune (Exp1). Ces nœuds sont de bons candidats pour la matérialisation.

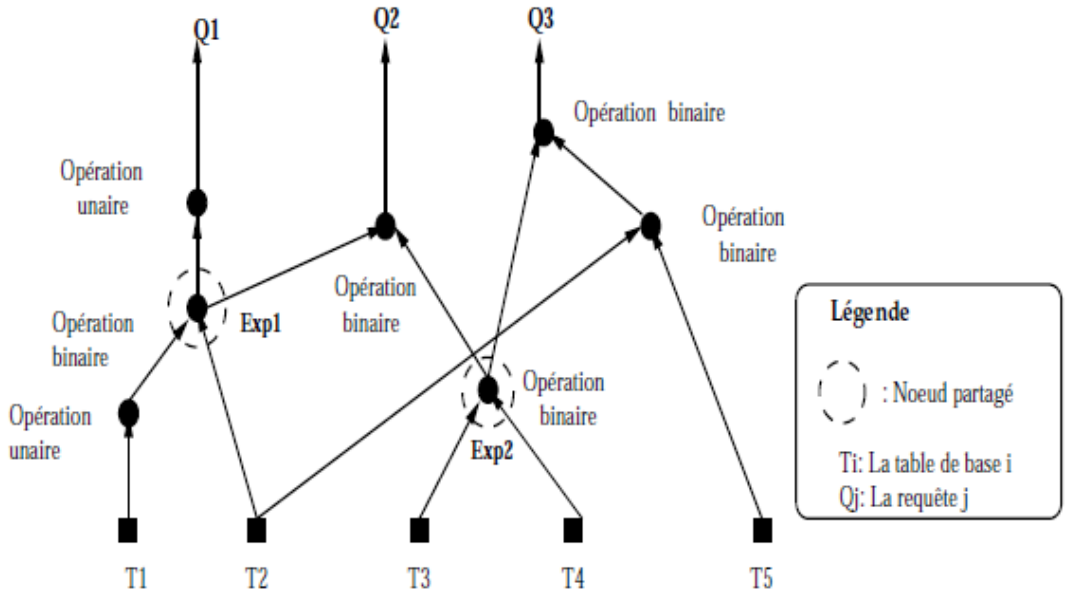


Figure 2.5 Le principe de base de la sélection de yang et al [64]

Les travaux de Baralis et al. [13] Dans le contexte MOLAP, Baralis et al. [13] ont développé une heuristique pour le PSV statique. Leur technique consiste à élaborer le treillis des vues qui prend en compte les hiérarchies des attributs

(par exemple jour → semaine → semestre).

Le treillis permet non seulement d'établir des dépendances entre les vue à matérialiser mais constitue également un bon support pour les algorithmes de sélection. Il permet aussi de dire dans quel ordre les vues sont matérialisées [53]. On dit qu'une vue V_i du treillis est une vue candidate si une des deux conditions suivantes est satisfaite :

– V_i est associée à quelques requêtes.

– Il existe deux vues candidates V_j et V_k, telle que V_i est la plus petite borne supérieure de $V_j$et V_k (le coût de mise à jour des vues V_j et V_k est supérieur à celui de V_i).

Une fois les vues candidates sélectionnées, le bénéfice pour chaque nœud descendant est recalculé et la vue de bénéfice maximal est sélectionnée.

Pour conclure sur ce type d'algorithmes, nous pouvons dire qu'étant donné l'absence de contrainte liée à ces algorithmes, il n'existe aucun moyen d'évaluer leur résultat [22].

4.2- Algorithmes dirigés par la contrainte d'espace

Dans le travail initial réalisé pour la sélection des vues, Harinarayan et al. [53] ont présenté un algorithme glouton pour sélectionner un ensemble de cellules (vues) dans un cube de données. L'objectif de cet algorithme est de minimiser le temps de réponse des requêtes sous la contrainte que la taille des vues sélectionnées ne dépasse pas la capacité d'espace S. Les auteurs s'intéressent seulement aux requêtes ayant des fonctions d'agrégation. Les auteurs ont modélisé le problème sous la forme d'un treillis de vues. Les auteurs ont développé un modèle de coût afin d'estimer le coût de chaque vue du treillis. Ce coût est basé sur le principe suivant :

Pour évaluer une requête Q (vue), nous choisissons une vue ancêtre de Q (disons Q_A), qui a été matérialisée. Le coût d'évaluation de la requête Q est le nombre de n-uplets présents dans la table correspondante à Q_A. Chaque vue est associée à un coût de stockage correspondant au nombre de n-uplets de cette vue.

L'algorithme de sélection des vues est décrit de la façon suivante :

Soit S l'ensemble des vues matérialisées à l'étape j de l'algorithme. Pour chaque vue v, un bénéfice dénoté par B(v, S) et défini comme suit :

– Pour chaque relation de dépendance $w \preceq v$, définir une quantité B_w par :

1. Soit u la vue ayant le plus petit coût dans S tel que $w \preceq u$
2. Si $C(v) < C(u)$, alors $B_w = C(u) - C(v)$, sinon, $B_w = 0$

– B (v, S) = $\sum_{w \leq v} B_w$

En d'autres termes, le bénéfice de V est calculé en considérant sa contribution dans la réduction du coût d'évaluation des requêtes. Pour chaque vue w couvrant v, nous calculons le coût d'évaluation w en utilisant v et d'autres vues dans S offrant un coût moins élevé pour évaluer w. Si la présence de la vue v est inférieure au coût de w, alors la différence représente une partie du bénéfice de la matérialisation de la vue v. Le bénéfice total B (v, S) est la somme de tous les bénéfices en utilisant v pour évaluer w, et fournissant un bénéfice positif. Les auteurs montrent que cet algorithme présente des performances très proches de l'optimum. Cependant, il parcourt l'espace des solutions possibles à un niveau élevé de granularité et peut éventuellement laisser échapper de bonnes solutions [1].

4.3- Algorithmes dirigés par le temps de maintenance

Ce type d'algorithmes a été étudié par Gupta [21]. Avant de décrire ces algorithmes, quelques définitions sont introduites.

Etant donné un graphe de vues de type ET-OU et une quantité S (temps de maintenance disponible), le PSV consiste à sélectionner un ensemble de vues minimisant le temps de réponse total tel que le temps total de maintenance des vues soit inférieur à S. Les auteurs ont présenté deux heuristiques pour résoudre ce problème :

(1) un algorithme "glouton" polynômial fournissant une solution quasi-optimale pour les graphes de vues de type ET et pour les graphes de vues de type OU (où chaque requête a de multiple plans d'exécution),

(2) un algorithme de type A^* pour les graphes de vues de type ET et OU. Notons que tout algorithme de type A^* cherche la solution optimale dans un graphe ayant un nombre petit de nœuds, où chacune représente une solution potentielle.

4.4- Les algorithmes évolutionnaires

Les algorithmes évolutionnaires emploient une stratégie de recherche aléatoire semblable à l'évolution biologique pour de bonnes solutions.

Bien qu'un algorithme évolutionnaire ressemble à des algorithmes aléatoires dans cet aspect, cette approche montre qu'assez de différences justifiaient une considération de ses propres. L'idée fondamentale est de commencer par une population initiale aléatoire et générer des descendants par les variations aléatoires (par exemple, croisement et mutation).Les

membres les plus convenables de la population survivent dans la sélection suivante ; la prochaine génération est basée sur ces derniers.

L'algorithme se termine dès qu'il n'y aura aucune autre amélioration sur une période ou après un nombre prédéterminé de générations.

Lee et le Hammer (2001) [65] se sont concentrés sur une solution efficace au problème de sélection de vue coût-maintenance en utilisant un algorithme génétique (l'un des algorithmes évolutionnaires) pour calculer ou trouver un ensemble prés-optimal de vues.. Dans cette étude, une solution est focalisée au problème de sélection de vue de coût de maintenance qui minimise le temps de réponse d'une requête en donnant des bornes variables supérieures sur le coût de maintenance, on assumant une quantité illimitée d'espace de stockage parce que l'espace de stockage n'est pas considéré cher et il n'est pas une ressource critique.

Un algorithme évolutionnaire contraint a été proposé [66]. Des contraintes sont incorporées à l'algorithme par un procédé de rang stochastique. Le rang stochastique, peut traiter des contraintes efficacement. L'algorithme proposé est capable de trouver une solution faisable proche de l'optimale et bien balancer avec la taille du problème.

D'abord des pools de chaîne binaire représentent des génomes sont générés aléatoirement, pour créer la population initiale, chaque génome représente une solution candidate au problème à résoudre. La longueur de ce génome est le nombre total de sommets dans le treillis;1 et 0 signifie que les sommets doivent être matérialisé ou pas, respectivement.

5- Conclusion

Nous avons décrit dans ce chapitre le contexte du problème de sélection des vues ainsi que les fonctions de coût à minimiser et les contraintes à respecter durant le développement des algorithmes.

Des formulations mathématiques ont été mentionné afin de les utiliser dans la résolution du problème, et enfin on a cité quelques algorithmes qui ont été proposés dans la littérature, et comme les algorithmes évolutionnaires tel que les algorithmes génétiques l'un des meilleurs métaheuristiques pour l'optimisation, nous avons donné un état de l'art de l'utilisation de ces algorithmes pour résoudre le PSV . La performance de cette approche nous a permis de les adopter dans notre étude. Dans le chapitre suivant nous détaillerons ces algorithmes.

1- Introduction

Le problème PSV a été à considéré comme un problème d'optimisation dans lequel on définit une ou plusieurs fonctions objectif, ou fonctions de coût, que l'on cherche à minimiser (ou maximiser) par rapport à l'ensemble des paramètres concernés. La définition du problème d'optimisation est souvent complétée par la donnée de contraintes : tous les paramètres des solutions retenues doivent respecter ces contraintes, faute de quoi ces solutions ne sont pas réalisables. La résolution d'un tel problème a conduit les chercheurs à proposer des méthodes de résolution de plus en plus performantes, parmi lesquelles on peut citer les métaheuristiques.

Dans ce chapitre, nous allons nous intéresser uniquement aux algorithmes génétiques qui ont connu ces quinze dernières années un développement considérable grâce à l'augmentation vertigineuse de la puissance des calculateurs. Le chapitre est organisé en deux parties. La première est consacrée aux algorithmes génétiques comme étant une méthode d'optimisation mono-objectif. La deuxième partie traite l'optimisation multi-objectifs par algorithmes génétiques. Nous commençons d'abord par un bref rappel sur les spécificités d'un problème à plusieurs objectifs, puis nous évoquons le concept de dominance et la notion d'optimalité au sens de Pareto qui caractérise cette classe de problèmes. Enfin, nous détaillons quelques algorithmes génétiques, dédiés à l'optimisation multi-objectifs tout en accordant plus d'importance à un algorithme élitiste de référence (NSGA-II) qui sera utilisé dans la suite de ce travail.

2- Les algorithmes génétiques :

Les algorithmes génétiques sont des algorithmes d'exploration fondés sur les mécanismes da la sélection naturelle et de la génétique. Ils utilisent à la fois les principes de la survie des structures les mieux adapté, et les échanges d'information pseudo-aléatoires, pour former un algorithme d'exploration qui possède de certaines des caractéristiques de l'exploration humaine [16].

2.1- Principe de fonctionnement :

Les algorithmes génétiques sont des algorithmes évolutionnaires, les individus soumis à l'évolution sont des solutions, plus ou moins performantes, à un problème posé. Ces solutions

appartiennent à l'espace de recherche du problème d'optimisation. L'ensemble des individus traités simultanément par l'algorithme évolutionnaire constitue une population. Elle évolue durant une succession d'itérations appelées générations jusqu'à ce qu'un critère d'arrêt, qui prend en compte a priori la qualité des solutions obtenues, soit vérifié. Durant chaque génération, une succession d'opérateurs est appliquée aux individus d'une population pour engendrer la nouvelle population à la génération suivante. Lorsqu'un ou plusieurs individus sont utilisés par un opérateur, on convient de les désigner comme des parents. Les individus résultant de l'application de l'opérateur sont des enfants. Ainsi, lorsque deux opérateurs sont appliqués en séquence, les enfants engendrés par l'un peuvent devenir des parents pour l'autre.
le fonctionnement d'un AG est alors basé sur les phases suivantes [60] :

1. **Initialisation** : une population initiale de taille N chromosomes est tirée aléatoirement.
2. **Evaluation** : chaque chromosome est décodé puis évalué.
3. **Reproduction:** création d'une nouvelle population de N chromosomes par l'utilisation d'une méthode de sélection appropriée.
4. **Opérateurs génétiques:** croisement et mutation de certains chromosomes au sein de la nouvelle population.
5. **Retour** à la phase 2 tant que la condition d'arrêt du problème n'est pas satisfaite.

2.2- Codage et population initiale

Premièrement, il faut représenter les différents états possibles de la variable dont on cherche la valeur optimale sous forme utilisable pour un AG: c'est le codage. Cela permet d'établir une connexion entre la valeur de la variable et les individus de la population, de manière à limiter la transcription génotype-phénotype qui existe dans le monde vivant. Il existe principalement trois types de codage : le codage binaire, le codage réel et le codage en base n.

2.2.1- Codage binaire

Ce codage a été le premier à être utilisé dans le domaine des AG. Il présente plusieurs avantages : alphabet minimum {0,1}, facilité de mise en point d'opérateurs génétiques et existence de fondements théoriques (théorie sur les schémas). Néanmoins ce type de codage présente quelques inconvénients :

1. Les performances de l'algorithme sont dégradées devant les problèmes d'optimisation de grande dimension à haute précision numérique. Pour de tels problèmes, les AG basés sur les chaînes binaires ont de faibles performances comme le montre michalewicz [41].
2. La distance de Hamming entre deux nombres voisins (nombre de bits différents) peut être assez grande dans le codage binaire : l'entier 7 correspond à la chaîne 0111 et la chaîne 1000 correspond à l'entier 8. Or la distance de hamming entre ces deux chaînes est de 4, ce qui crée bien souvent une convergence, et non pas l'obtention de la valeur optimale.

2.2.2- Codage réel

Il a le mérite d'être simple. Chaque chromosome est en fait un vecteur dont les composantes sont les paramètres du processus d'optimisation. Par exemple, si on recherche l'optimum d'une fonction de n variables f(x1, x2, . . . , xn−1, xn), on peut utiliser tout simplement un chromosome *ch* contenant les n variables: Avec ce type de codage,

ch: x1 x2 · · · xn−1 xn

La procédure d'évaluation des chromosomes est plus rapide vu l'absence de l'étape de transcodage (du binaire vers le réel). Les résultats donnés par Michalewicz [41] montrent que la représentation réelle aboutit souvent à une meilleure précision et un gain important en termes de temps d'exécution.

2.2.3- Codage en base n

Dans ce type de codage, les gènes constituant un chromosome sont des chiffres exprimés dans une base de numération n, ce qui permet de représenter n valeurs discrètes.
L'AG démarre avec une population composée de N individus dans le codage retenu. Le choix des individus conditionne fortement la rapidité de l'algorithme. Si la position de l'optimum dans l'espace de recherche est totalement inconnue, il est intéressant que la population soit répartie sur tout l'espace de recherche. Si par contre des informations à priori sur le problème sont disponibles, il paraît évident de générer les individus dans un espace particulier afin d'accélérer la convergence. Disposant d'une population initiale souvent non homogène, la diversité de la population doit être entretenue aux cours des générations afin d'explorer le plus

largement possible l'espace de recherche. C'est le rôle des opérateurs de croisement et de mutation.

2.3- Opérateur de croisement

Le croisement est le principal opérateur agissant sur la population des parents. Il permet de créer de nouveaux individus par l'échange d'information entre les chromosomes par leur biais de leur combinaison. La population courante est divisée en deux sous populations de même taille (N/2) et chaque couple formé par un membre provenant de chaque sous population participe à un croisement avec une probabilité (pc) souvent supérieure à 0,5. Si le croisement a eu lieu entre deux chromosomes parents (ch_1 et ch_2), constitués de ***l*** gènes, on tire aléatoirement une position de chacun des parents. On échange ensuite les deux sous chaînes terminales de chacun des chromosomes, ce qui produit deux enfants ch_1' et ch_2' comme indiqué sur la figure 3.1.

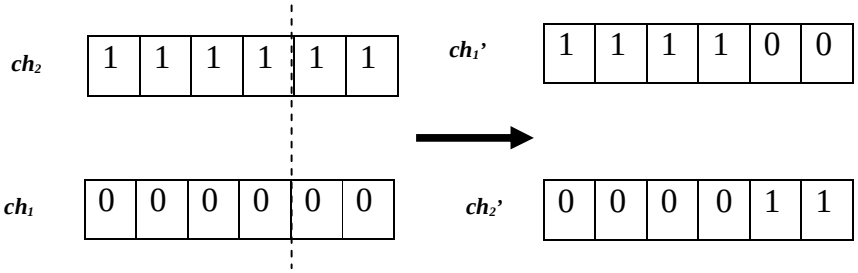


Figure 3.1 Croisement standard en un seul point

Dans notre exemple, Figure 3.1, un croisement localisé à la quatrième position a eu lieu entre les chromosomes ch_1 et ch_2: il s'agit bien d'un croisement en un seul point.

Ainsi on peut étendre ce principe de combinaison en choisissant non pas un seul point, mais 2, 3, etc, (croisement en multipoints) [38]. Sur la figure 3.2 nous représentons un croisement en deux points,

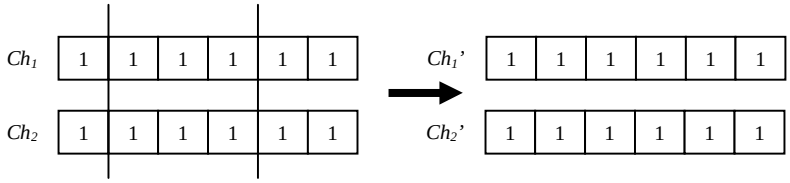


Figure 3.2 Croisement standard en deux points

Ce type de croisement est très efficace et peut s'étendre à n'importe quel type de chaînes (réelles, en base n, etc...). Néanmoins certains auteurs [56] préfèrent utiliser dans le cas des chaînes réelles, un croisement de type barycentre: deux gènes $ch_1(i)$ et $ch_2(i)$ sont sélectionnés dans chacun des parents à la même position *i*. Ils définissent deux nouveaux gènes $ch_1'(i)$ et $ch_2'(i)$ par combinaison binaire :

$$\begin{cases} ch'_1(i) = \alpha \times ch_1(i) + (1-\alpha) \times ch_2(i) \\ ch'_2(i) = (1-\alpha) \times ch_1(i) + \alpha \times ch_2(i) \end{cases} \quad (1.1)$$

où α est un paramètre de pondération aléatoire qui prend généralement ses valeurs dans l'intervalle [−0.5, 1.5] (ceci permet de générer des points entre ou à l'extérieur des deux gènes considérés).

Quoi qu'il en soit, il se peut que l'effet de l'opérateur de croisement soit insuffisant pour assurer une meilleure exploration de l'espace de recherche. Ainsi dans le cas du codage binaire, certaines chaînes peuvent totalement disparaître de la population. Par exemple, si aucun chromosome de la population initiale ne contient de 1 en première position et que ce 1 fasse partie de la chaîne optimale à trouver, aucun croisement ne peut faire apparaître cet élément. Ce dernier ne peut s'introduire dans la population que si l'on introduit un autre opérateur et c'est, entre autre, pour remédier à ce problème que l'opérateur de mutation est utilisé.

2.4- Opérateur de mutation

Le rôle de cet opérateur est de modifier aléatoirement la valeur d'un gène dans un chromosome. Dans le cas du codage binaire, chaque bit $a_i \in \{0,1\}$ est remplacé par son complémentaire $\bar{a}_i = 1 - a_i$. Dans l'exemple de la figure 3.3, une mutation a eu lieu sur le deuxième gène du chromosome *ch* et elle a transformé ce gène de 1 en 0.

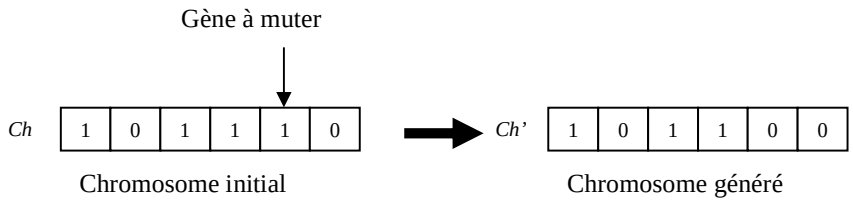


Figure 3.3 Principe de l'opérateur de mutation

Pour les chaînes codées en base *n*, la mutation consiste à remplacer le gène initial par un chiffre en base *n* tiré au sort.

Dans le cas d'un codage réel, on utilise principalement deux opérateurs de mutation: la mutation *uniforme* et la mutation *non uniforme* [41]. En supposant fixée la probabilité de mutation p_m, un tirage au sort pour chaque gène x_k d'un chromosome *ch* permet de décider si ce gène doit être ou non modifié. Nous supposons que le gène prend ses valeurs dans un intervalle$\left[x_k^{min}, x_k^{max}\right]$.

Pour la mutation uniforme, qui est une simple extension de la mutation binaire, on remplace le gène x_k sélectionné par une valeur quelconque x_k' k tirée aléatoirement dans l'intervalle $\left[x_k^{min}, x_k^{max}\right]$.

Pour la mutation non uniforme, le calcul de la nouvelle valeur d'un gène est un peu plus complexe. Le gène x_k subit des modifications importantes durant les premières générations, puis graduellement décroissantes au fur et à mesure que l'on progresse dans le processus d'optimisation. Pour une génération *t*, on tire au sort une valeur binaire qui décidera si le changement doit être positif ou négatif. La nouvelle valeur x_k' du gène x_k est donnée par :

$$x_k' = \begin{cases} x_k + \Delta(t, x_k^{max} - x_k) & si\ rand = 0 \\ x_k - \Delta(t, x_k - x_k^{min}) & si\ rand = 0 \end{cases} \qquad (1.2)$$

où $\Delta(t,y)$ est une fonction qui définit l'écart entre la nouvelle valeur et la valeur initiale à la génération *t* et *rand* est nombre aléatoire qui prend les valeurs 0 ou 1.

Dans [28], les auteurs proposent d'utiliser une fonction $\Delta(t,y)$ correspondante à une décroissance exponentielle de l'écart à travers les générations. Cette fonction est définie par :

$$\Delta(t, y) = y \times (1 - r^{(1-\frac{t}{T})^{\beta}}) \quad (1.3)$$

où *T* est l'indice de génération pour laquelle l'amplitude de la mutation s'annule, β est un paramètre de l'opérateur de mutation (souvent $\beta = 5$), *r* est un nombre produit aléatoirement dans l'intervalle [0,1] et *t* le numéro de la génération.

Si par contre, l'intervalle de variations du gène x_k n'est pas connu, une mutation gaussienne est souvent utilisée. Le principe de base de ce type de mutation est d'ajouter un bruit gaussien centré *N(0,σ)* au gène que l'on désire faire muter:

$$x'_k = x_k + N(0,\sigma) \qquad (1.4)$$

où σ représente la variance.

La mutation est traditionnellement considérée comme un opérateur intervenant à la marge, dans la mesure où sa probabilité est en général assez faible (de l'ordre de 1%). Mais elle confère aux algorithmes génétiques une propriété très importante : l'ergodicité (tous les points de l'espace de recherche peuvent être atteints). Cet opérateur est donc d'une grande importance et il est loin d'être marginal, d'ailleurs un algorithme génétique peut converger sans l'opérateur de croisement et certaines applications fonctionnent de cette manière [28].

2.5- Fonction d'évaluation

Un algorithme génétique nécessite généralement la définition d'une fonction rendant compte de la pertinence des solutions potentielles, à partir des grandeurs à optimiser.

Nous la nommerons fonction d'adaptation *f* (ou fitness function en terminologie anglosaxonne). Elle est souvent exprimée par la composition de deux fonctions *g* et *h*:

$$f = g \circ h \qquad (1.5)$$

avec:

$\circ$: loi de composition de fonctions.

g: fonction de transformation, pouvant être linéaire, exponentielle, logarithmique, etc . . ..

h: fonction objectif ou de coût, elle dépend de l'objectif recherché.

2.6- Opérateur de sélection

La sélection crée une population intermédiaire constituée de copies des individus de la population courante. En règle générale, le nombre de copies d'un individu est lié directement à la fitness relative de l'individu au sein de la population. Il existe plusieurs méthodes heuristiques qui représentent la reproduction, la méthode la plus connue et la plus utilisée est la sélection par roulette biaisée (roulette wheel selection) de Goldberg [16]. Selon cette méthode,

chaque chromosome est copié dans la nouvelle population proportionnellement à sa fitness. On effectue en quelque sorte, autant de tirages avec remise que d'éléments existant dans la population. Ainsi pour un chromosome particulier ch_i de fitness f (ch_i), la probabilité de sa sélection dans la nouvelle population de taille N est :

$$p(ch_i)\frac{f(ch_i)}{\sum_{j=1}^{N} f(ch_j)} \qquad (1.6)$$

Plus la performance d'un individu est élevée par rapport à celle des autres, plus il a une chance d'être reproduit dans la population. Les individus ayant une grande fitness relative ont donc plus de chance d'être sélectionnés. On parle alors de sélection proportionnelle.

Le nombre de copies espérées pour chaque individu ch_i qui va résulter de la sélection est alors égal à :

$$n_i = N \times p(ch_i) = \frac{f(ch_i)}{\frac{1}{n}\sum_{j=1}^{N} f(ch_j)} \qquad (1.7)$$

L'inconvénient majeur de ce type de reproduction vient du fait qu'il peut favoriser la dominance d'un individu qui n'est pas forcément le meilleur. Cette méthode peut aussi engendrer une perte de diversité par la dominance d'un super-individu. Pour palier cet inconvénient, on préfère souvent des méthodes qui n'autorisent en aucun cas l'apparition de super-individu. Par exemple, la sélection par tournoi (tournement selection) ou d'autres méthodes faisant intervenir un changement d'échelle (Scalling) et/ou des notions de voisinage entre chromosomes (Sharing).

2.6.1- Sélection par tournoi

On tire deux individus aléatoirement dans la population et on reproduit le meilleur des deux dans la nouvelle population. On répète la procédure jusqu'à ce que la nouvelle population soit complète.

2.6.2- Sharing

Le *sharing* consiste à ajuster la fitness des individus pour éviter qu'ils se concentrent dans une niche principale (optimum globale). La technique de partage de la fitness (fitness *sharing*), introduite par Goldberg et Richardson [15], réduit la fitness de chaque individu d'un facteur correspondant environ au taux d'agrégation de la population autour de son voisinage :

$$f'(ch_i) = \frac{f(ch_i)}{m_i} \qquad (1.8)$$

Où le compteur de niche m_i se calcule de la manière suivante:

$$m_i = \sum_{j=1}^{N} sh(d_{ij}) \qquad (1.9)$$

Où N désigne la taille de la population et sh mesure la similarité entre deux individus i et j en fonction de la distance d_{ij} et le rayon de niche σ_{shar} :

$$sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{shar}}\right)\alpha & si\ d_{ij} < \sigma_{shar} \\ 0 & sinon \end{cases} \qquad (1.10)$$

Si le principe de *sharing* a été utilisé initialement dans l'espace de paramètres (le critère de distance étant fonction des paramètres), il est tout a fait possible de l'adapter dans l'espace des objectifs. C'est d'ailleurs cette idée qui est exploitée dans les algorithmes génétiques multi-objectifs.

2.6.3- Elitisme

La stratégie élitiste consiste à conserver le meilleur individu à chaque génération. Ainsi l'élitisme empêche l'individu le plus performant de disparaître au cours de la sélection ou que ses bonnes combinaisons soient affectées par les opérateurs de croisement et de mutation. Après chaque évaluation de la performance des individus à une génération t donnée, le meilleur individu de la génération précédente *(t−1)* est réintroduit dans la population si aucun des individus de la génération t n'est meilleur que lui. Par cette approche, la performance du

meilleur individu de la population courante est monotone de génération en génération. Il apparaît que l'élitisme améliore considérablement les performances de l'algorithme génétique pour certaines classes de problèmes, mais peut les dégrader pour d'autres classes, en augmentant le taux de convergences prématurées.

3- Optimisation multi-objectifs

Dans la section précédente, nous avons considéré uniquement le cas où le problème à traiter possédait un objectif unique à optimiser. En pratique ces problèmes sont rares, il s'agit souvent de satisfaire plusieurs critères simultanément. L'optimisation multi-objectifs s'intéresse à ce type de problème que l'on peut définir de la manière suivante :

$$\begin{cases} \min(F(X) = (f_1(X),\ f_2(X) \dots f_n(X))) \\ X \in C \end{cases} \qquad (1.11)$$

où *n* est le nombre de fonctions objectifs, $X = [x_1,x_2,\dots x_m]$ est le vecteur représentant les variables de décision. C représente l'ensemble des solutions réalisables associé à des contraintes d'égalité et d'inégalité et $F(X) = (f_1(X),f_2(X)\dots f_n(X))$ est le vecteur d'objectifs. Dans un problème d'optimisation multi-objectifs, il y a plus qu'une fonction d'objectif $(n \geq 2)$, chaque fonction peut avoir un optimum différent. Le but d'un problème multi-objectifs est donc de trouver de "bons compromis" plutôt qu'une seule solution (à moins qu'une solution soit optimale pour toutes les fonctions objectifs, ce qui est rarement le cas). Lorsqu'il y a plusieurs objectifs, la notion d'optimum change, elle est remplacée par les notions de dominance et d'optimalité de Pareto.

Définition 1 (la dominance): une solution A domine une solution B pour un problème de minimisation (resp. maximisation) si et seulement si:

$$\forall i \in \{1,2,\dots,n\}:\ f_i(A) \leq f_i(B) (resp.\ f_i(A) \geq f_i(B))$$
$$et\ \exists j \in \{1,2,\dots,n\}:\ f_j(A) < f_j(B) (resp.\ f_j(A) > f_j(B))$$

On dit que B est dominée par A ou entre les deux solutions, A est la solution non dominée.

Définition 2 (Pareto optimum): un vecteur $X^* \in C$ est un optimum de Pareto s'il n'existe aucune solution X de C qui domine X^*.

Au lieu d'une solution unique, l'optimisation multi-objectifs donne lieu à un ensemble de solutions optimales. Toute solution de cet ensemble est optimale dans le sens qu'il est impossible d'améliorer les performances, sur un critère de cette solution, sans que cela entraîne une dégradation des performances, sur au moins un autre critère. Ces solutions optimales forment l'ensemble de solutions Pareto optimales, elles sont aussi connues sous le nom de *solutions efficaces, non inférieures* et *non dominées.* La représentation de ces solutions non dominées dans l'espace d'objectif est appelée front de Pareto. La figure 3.4 montre un exemple de front de Pareto pour un problème de minimisation à deux objectifs.

L'ensemble de points blancs représentent le front de Pareto. Les algorithmes génétiques, avec un bon réglage de leurs paramètres, constituent une approche intéressante pour la résolution des problèmes d'optimisation multi-objectifs.

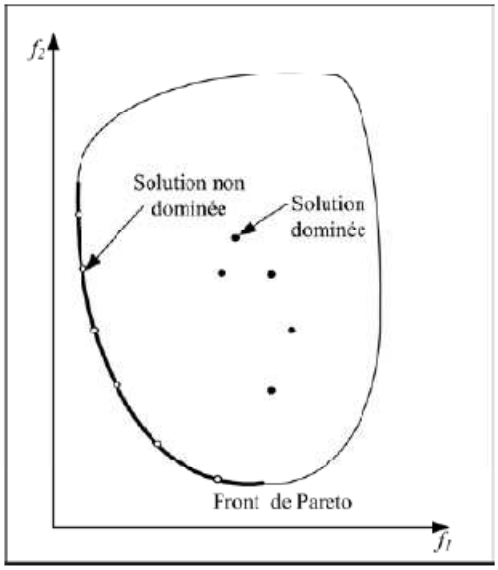


Figure 3.4 Front de Pareto [61]

De plus, ce domaine est très dynamique et ne cesse de se développer. Il a été proposé plusieurs méthodes pour le traitement des problèmes multi-objectifs. Ces méthodes peuvent être classées principalement en deux catégories: La première catégorie enveloppe les méthodes "agrégatives", qui transforment le problème en un problème uni-objectif utilisant une fonction objectif équivalente.

La deuxième famille comporte les méthodes dites "non agrégatives", dans ces méthodes il n'y a pas fusion d'objectifs pour se ramener à un problème d'optimisation mono-objectif.

3.1- Méthode agrégative

C'est l'une des premières méthodes utilisée pour résoudre les problèmes d'optimisation multi objectifs (MO). Elle consiste à transformer le problème MO en un problème mono objectif en combinant les composantes f_i du vecteur objectif du problème en une seule fonction scalaire *f*. Il existe dans la pratique, différentes façons de construire la fonction *f*. La plus classique et la plus utilisée se ramène à une simple somme pondérée des objectifs f_i (agrégation additive):

$$f = \sum_{i=1}^{n} \lambda_i \times f_i$$

où les paramètres λ_i sont les poids de pondération. La figure 3.5 illustre l'interprétation géométrique de la méthode de pondération dans le cas d'un problème à deux objectifs. Fixer un vecteur de poids revient à trouver un hyper plan dans l'espace objectif (une ligne pour un problème bi-objectif). La solution Pareto optimale est le point où l'hyper-plan possède une tangente commune avec l'espace réalisable. La méthode de pondération est relativement simple à utiliser mais il est assez difficile de fixer a priori les valeurs des poids associés à chaque objectif. Il est par ailleurs impératif de normaliser les objectifs lorsque ceux-ci sont non-commensurables, ce qui est souvent le cas. Ce problème de mise à l'échelle s'avère complexe car les valeurs de normalisation ne sont pas faciles à déterminer. De plus, comme le montre la figure 3.5-b, cette méthode n'est pas adaptée aux espaces non convexes qui empêchent l'algorithme de converger vers l'optimum global. Le champ applicatif de cette méthode se trouve alors fortement réduit.

Grâce au principe de pondération, il est possible avec un algorithme génétique standard d'obtenir le front de Pareto d'un problème multi-objectifs. Il est nécessaire pour cela d'exécuter l'algorithme plusieurs fois successives en faisant varier les facteurs de pondération (voir figure 3.6).

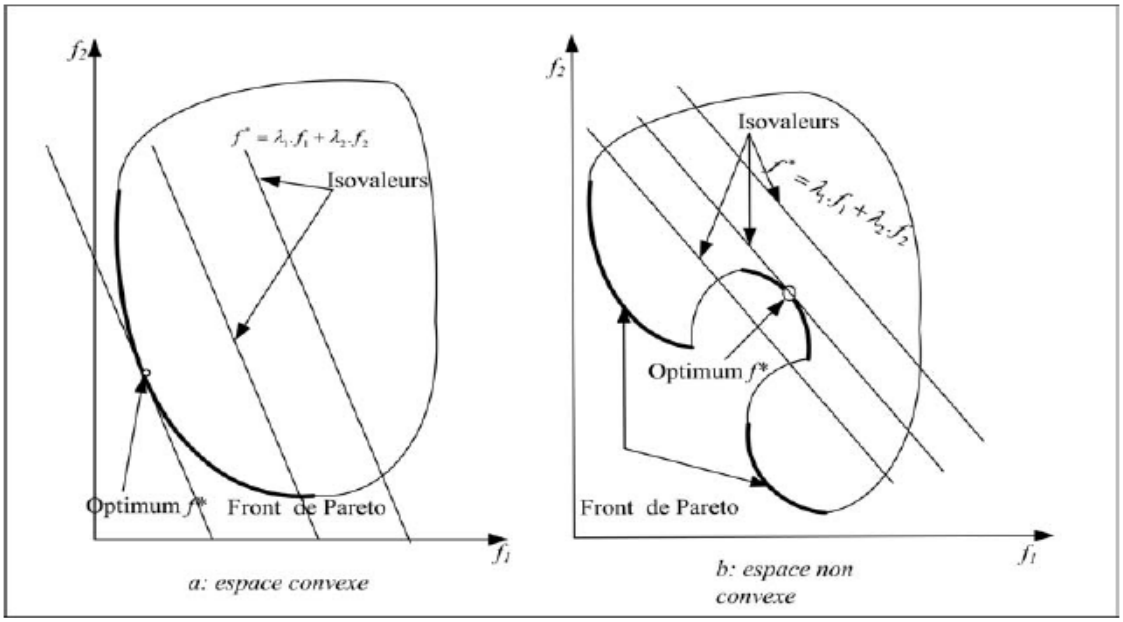


Figure 3.5 Interprétation géométrique de la méthode d'agrégation [61]

Toutefois, une variation uniforme des poids ne garantit pas la détermination des solutions Pareto optimales uniformément réparties sur le front.

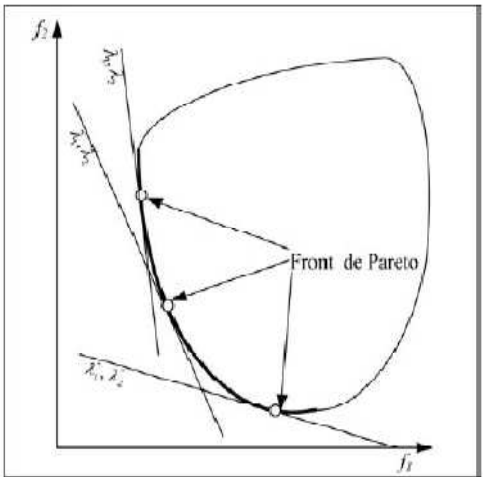


Figure 3.6 Illustration de la variation des coefficients de pondération pour la recherche du front de Pareto [61].

3.2- Méthode non agrégative

Comme nous l'avons signalé précédemment, la méthode agrégative peut être utilisée de façon séquentielle pour obtenir le front de Pareto pour un problème d'optimisation multi-objectifs. Toutefois, cette approche n'est généralement pas satisfaisante car le nombre

d'exécutions successives nécessaires pour déterminer les différents compromis conduit alors à un nombre d'évaluations de critères prohibitif. Ainsi, pour surmonter cette difficulté, on préfère utiliser des méthodes permettant d'une part de trouver l'ensemble de solutions Pareto optimales en une seule exécution et d'autre part de s'affranchir des problèmes de mise à l'échelle des objectifs. Parmi ces méthodes on peut citer :

– Vector Evaluated Genetic Algorithm (VEGA)

– Multiple Objective Genetic algorithm (MOGA)

– Niched Pareto genetic algorithm (NPGA)

– Nondominated sorting genetic algorithm (NSGA) et sa deuxième version (NSGAII)

3.2.1- Vector evaluated genetic algorithm (VEGA)

Le VEGA (Vector evaluated genetic algorithm) proposé par Schaffer (1985), a été la première méthode non agrégative utilisant les algorithmes génétiques pour résoudre un problème d'optimisation multi-objectif. Cet algorithme considère une population de N individus. A chaque génération, la population est divisée en un nombre de sous populations égal au nombre d'objectifs. Chaque sous population i est sélectionnée en considérant un seul objectif f_i. Ensuite, ces sous populations sont regroupées afin d'obtenir une nouvelle population de N individus et les opérateurs de croisement et de mutation sont appliqués. L'avantage de cet algorithme est qu'il est facile à implémenter et à combiner avec n'importe quel mode de sélection (tournoi, roulette, rang), mais son inconvénient majeur est qu'il a tendance à générer des solutions qui excellent dans un seul objectif, sans tenir compte des autres objectifs (points extrêmes du front). Toutes les solutions de performance moyenne (ne possédant aucun objectif fort) et qui peuvent être de bons compromis, risquent de disparaître avec ce type de sélection.

3.2.2- Multiple Objectives Genetic Algorithm (MOGA)

Cet algorithme, proposé par Fonseca et Fleming (1993), utilise la notion de dominance pour ranger les individus de la population. Il diffère de l'algorithme génétique standard uniquement dans la manière dont la fitness est assignée pour chaque solution. Pour démarrer l'algorithme, les relations de domination sont d'abord calculées pour chaque solution. Puis, pour une solution i, un rang égal à un plus le nombre de solutions ni qui dominent la solution i

est attribué. Une fitness est ensuite attribuée à chaque solution en fonction de son rang, les individus avec les rangs les plus faibles ayant les meilleures fitness. Afin de maintenir la diversité entre les solutions non dominées, les auteurs utilisent une fonction de partage (*Sharing*). La méthode permet d'obtenir des solutions de bonne qualité et s'implante facilement. Toutefois, les performances sont très dépendantes de la valeur du paramètre σ_{shar}utilisé dans le *sharing*.

3.2.3- NSGA (Nondominated Sorting Genetic Algorithm)

Dans l'algorithme NSGA proposé par Srinivas et Deb (1993), le calcul de fitness s'effectue en divisant d'abord la population en plusieurs fronts en fonction du degré de dominance au sens de Pareto de chaque individu. Les individus non dominés de la population courante constituent le premier front de Pareto. On attribue alors à tous les individus de ce front la même valeur de fitness factice. Cette valeur est supposée donner une chance égale de reproduction à tous ces individus. Mais pour maintenir la diversité de la population, il est nécessaire d'appliquer une fonction de partage sur cette valeur. Ensuite, ce premier groupe d'individus est temporairement supprimé de la population. On recommence cette procédure jusqu'à l'identification des solutions du deuxième front. La valeur factice de fitness attribuée à ce second groupe est inférieure à la plus petite fitness, après application de la fonction de partage sur le premier front. Ce mécanisme est répété jusqu'à ce que l'on ait traité tous les individus. L'algorithme se déroule ensuite comme un algorithme génétique standard. Grâce à sa procédure d'assignement de fitness basée à la fois sur la notion de dominance et la fonction de partage, le NSGA semble le plus approprié à maintenir la diversité de la population et à répartir plus efficacement les solutions sur le front de Pareto. Néanmoins, cet algorithme présente quelques insuffisances en raison de sa complexité de calcul et de sa sensibilité au choix de la valeur σ_{shar}.

3.2.4- Niched Pareto Genetic Algorithm (NPGA)

Cette méthode proposée par Horn et Napfliotis (1994) utilise une sélection par tournoi en se basant sur la notion de dominance de Pareto. Le NPGA exécute les mêmes étapes que l'AG standard, la seule chose qui diffère étant la méthode de sélection. A chaque tournoi, deux

individus candidats A et B sont pris au hasard dans la population courante. Au lieu de limiter la comparaison aux deux individus (comme c'est le cas pour l'AG standard), une sous population (ou ensemble de comparaison) de taille t_{dom} est également choisie au hasard. Les deux candidats sélectionnés sont comparés à chaque individu du sous-groupe. Si l'un des candidats est dominé par l'ensemble de comparaison et le second ne l'est pas, ce dernier est alors positionné dans la population suivante. Dans les autres cas, une fonction de partage est appliquée pour choisir le candidat gagnant. Le paramètre t_{dom} permet de contrôler la pression de sélection ou de dominance.

L'algorithme NPGA est considéré comme étant l'algorithme le plus rapide parmi les approches précédentes car à chaque génération la comparaison n'est appliquée que sur une portion de la population. Le principal inconvénient de cet algorithme est qu'il nécessite, en plus de spécifier le paramètre de *sharing* σ_{shar}, un autre paramètre supplémentaire qui est la taille du tournoi t_{dom}.

3.2.5- NSGA-II

Toutes les méthodes que nous venons de présenter ne conservent pas leurs solutions Pareto-optimales trouvées au cours des générations. Elles sont dites non élitistes. Pour résoudre cette difficulté, de nouvelles techniques ont été appliquées. Nous avons choisi de présenter uniquement le NSGA-II car il a été utilisé dans nos travaux. Nous ne ferons que citer les deux autres méthodes les plus connues à savoir Strength Pareto Evolutionary Algorithm (SPEA) et Pareto Envelope-based Selection Algorithm (PESA) présentées plus amplement dans les références [57] et [8].

En proposant le NSGA II, Deb [9] a tenté de résoudre toutes les critiques faites sur NSGA: non élitiste, complexité de calcul et utilisation de sharing qui implique le réglage d'un ou plusieurs paramètres. Dans cet algorithme, à chaque génération *t* une population de parents (P_t) de taille *N* et une population d'enfants (Q_t) de même taille sont assemblées pour former une population (R_t) de taille 2N, comme indiqué sur la figure 3.7. Cet assemblage permet d'assurer l'élitisme. La population (R_t) est ensuite répartie en plusieurs fronts (F_1, F_2, . .) par une procédure de tri, plus rapide que celle proposée dans la première version de NSGA. Une nouvelle population parent (P_{t+1}) est formée en ajoutant les fronts au complet (premier front F_1, second front F_2,

etc...) tant que ceux-ci ne dépassent pas *N*. Si le nombre d'individus présents dans (P_{t+1}) est inférieur à *N*, une procédure de crowding est appliquée sur le premier front suivant F_i non inclus dans (P_{t+1}). Le but de cet opérateur est d'insérer les (N − P_{t+1}) meilleurs individus de F_i qui manquent dans la population (P_{t+1}). Une fois que les individus de la population (P_{t+1}) sont identifiés, une nouvelle population enfant (Q_{t+1}) est créée par sélection, croisement et mutation. La sélection par tournoi est utilisée mais le critère de sélection est maintenant basé sur l'opérateur de comparaison (<n) défini ci-dessous. Le processus se répète d'une génération à une autre jusqu'à satisfaction d'un critère d'arrêt.

Procédure de tri rapide (Fast sorting non dominated)

La répartition de la population en plusieurs fronts s'effectue de la manière suivante :

1. Pour chaque solution *p* de R_t on calcule deux paramètres:

– n_p (compteur de dominance), il représente le nombre de solutions qui dominent la solution *p*

– S_i ensemble de solutions dominées par *p*.

2. $i = 1$ initialisation du compteur de front

3. Identification des solutions non dominées $n_p = 0$, ces solutions forment le front F_i.

4. Pour chaque solution *p* de F_i on parcourt l'ensemble S_p et on retranche 1 au n_p de chaque solution.

5. $i = i + 1$ incrémente le compteur de front

6. On recommence les étapes à partir de 3 jusqu'à ce que tous les points soient traités.

Cet algorithme est d'une complexité de $O(k.N^2)$, alors que celui utilisé dans la première version est de $O(k.N^3)$. *k* étant la taille du vecteur objectifs

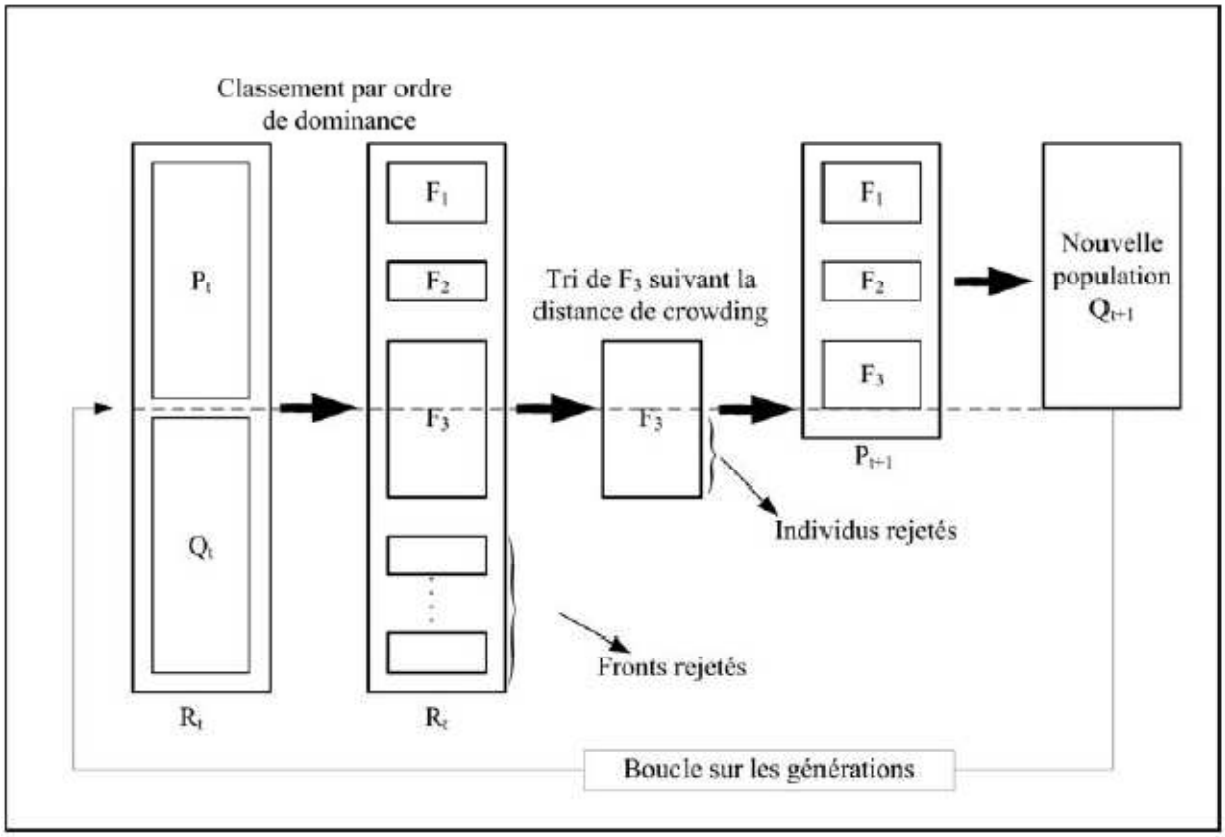


Figure 3.7 schéma de l'évolution de l'algorithme NSGA-II [61]

Distance de crowding

La dernière critique faite sur le NSGA est l'utilisation du sharing. Une méthode qui exige le réglage d'un ou plusieurs paramètre(s). Dans NSGA-II, Deb et al remplacent la procédure de *sharing* par une procédure de *crowding*, basée sur un calcul de distance (distance de *crowding*) qui ne nécessite aucun paramétrage et qui est également d'une complexité algorithmique moindre que celle de *sharing*. La distance de *crowding* d'une solution particulière *i* se calcule en fonction du périmètre de l'hypercube ayant comme sommets les points les plus proches de *i* sur chaque objectif. Sur la figure 3.8, est représenté l'hypercube en deux dimensions associé au point i. Le calcul de la distance de *crowding* nécessite, avant tout, le tri des solutions selon chaque objectif, dans un ordre ascendant. Ensuite, pour chaque objectif, les individus possédant des valeurs limites se voient associés une distance infinie. Pour les autres solutions intermédiaires, on calcule une distance de *crowding* égale à la différence normalisée des valeurs de fonctions objectives de deux solutions adjacentes. Ce calcul est

réalisé pour chaque objectif. La distance de *crowding* d'une solution est obtenue en sommant les distances correspondantes à chaque objectif.

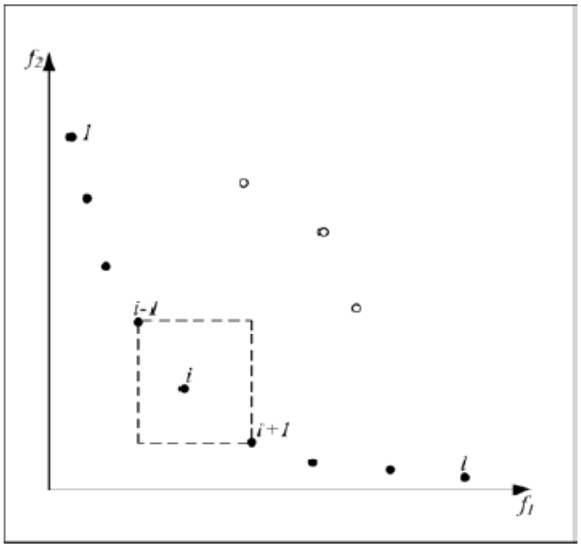


Figure 3.8 Distance de crowding (les points noirs sont des solutions appartenant au même front) [61]

1 : Algorithm 1 Calcul de la distance de crowding pour chaque solution d'un front

1. $l = |i\ \ |$, nombre de solution dans le front I
2. Pour chaque solution i poser $[i]_{distance} = 0$, Initialisation des distances.
3. Pour chaque objectif *m* :
    - $I = trier\ (I, m)$, trier *I* par ordre croissant selon le critère *m*.
    - $I[1]_{distance} = I[l]_{distance} = +\infty$
    - Pour i = 2 jusqu'à $l - 1$ faire

$$I[i]_{distance} = I[i]_{distance} + (\frac{f_m^{i+1} - f_m^{i-1}}{f_m^{Max} - f_m^{Min}})$$

Dans cet algorithme, f_m^{i+1}et f_m^{i-1} représentent respectivement les valeurs de la *m*ième fonction objectif des solutions *i + 1* et *i − 1* alors que les paramètres f_m^{max} et f_m^{min} désignent les valeurs maximale et minimale de la *m*ième fonction objectif.

Opérateur de crowding de comparaison ***($\prec_n$)***

Cet opérateur est utilisé pour guider le processus de sélection comme suit : chaque solution *i* de la population est identifiée par son rang i_{rank} et sa distance de crowding $i_{distance}$. L'opérateur $\prec_n$, défini ci-dessous, permet d'établir un ordre de préférence entre deux solutions:

$$i \prec_n j \quad \text{si} \quad (i_{rank} < j_{rank})$$

$$\text{ou } (i_{rank} = j_{rank}) \text{ et } (i_{distance} > j_{distance})$$

Entre deux solutions de fronts différents, on préfère la solution avec le plus petit front. Pour deux solutions qui appartiennent au même front, on préfère la solution située dans une région dépeuplée, c'est-à-dire la solution possédant la plus grande valeur de distance de crowding.

Boucle principale de l'algorithme NSGA-II

L'algorithme 3.1, représente la boucle principale du NSGA-II. Initialement, il y a la création aléatoire d'une population parent P_0. Chaque individu de cette population est affecté à un front de Pareto adéquat. On applique les opérateurs génétiques (sélection, croisement et mutation) pour générer la population enfant Q_0 de taille N à partir de la population P_0. L'élitisme est assuré par la comparaison de la population courante P_t avec la population précédente P_{t-1}.

Début
$\boldsymbol{R_t} := \boldsymbol{P_t} \cup \boldsymbol{Q_t}$ /*Combinaison des pop. Parent et enfant $|R_t| = 2N$ */
$\boldsymbol{F} := \boldsymbol{Fast-nondominated-sort}\ (\boldsymbol{R_t})$ /* F= $\{F_1, F_2 \ldots\ldots\}$ ensemble des Front de Pareto tel que $\sum|F_i| = 2N$ */
$\boldsymbol{p_{t+1}} := \emptyset$ /*initialisation*/
$\boldsymbol{i} := \boldsymbol{1}$
Tant que $|\boldsymbol{P_{t+1}}| + |\boldsymbol{F_i}| \leq \boldsymbol{N}$ /* jusqu'à la taille max de la pop . N */
Crowding-distance-assignment $(\boldsymbol{F_i})$ /* Calcul de la distance « crowded» dans Front F_i */
$\boldsymbol{P_{t+1}} := \boldsymbol{P_{t+1}} \cup \boldsymbol{F_i}$ /*Reconstruction de la population parent en ajoutant i^{ème} front de pareto
$\boldsymbol{i} := \boldsymbol{i+1}$ /* incrémentation vers le front suivant*/
Fin tanque
Sort $(\mathbf{F_i}, \prec_n)$ /* Triage des solutions F_i dans l'ordre décroissant suivant $\prec_n$
$\boldsymbol{P_{t+1}} := \boldsymbol{P_{t+1}} \cup \boldsymbol{F_i}\ [\boldsymbol{1}: (\boldsymbol{N} - |\boldsymbol{P_{t+1}}|)]$ /*Choisir juste les premiers éléments F_i pour compléter la population parent P_{t+1} jusqu'à N individus */
$\boldsymbol{Q_{t+1}} := \boldsymbol{make-new-pop}(\boldsymbol{P_{t+1}})$ /* Application des opérateurs génétique (sélection, croisement, mutation) pour la génération d'une nouvelle pop. Q_{t+1} */
$\boldsymbol{t} := \boldsymbol{t+1}$ /* génération suivante */
fin

Algorithme 3.1 –Boucle principale NSGA-II [60]

3.3- Convergence et diversité des solutions des algorithmes multi-objectifs

Au cours de ces dernières années, la recherche sur des algorithmes évolutionnaires a démontré leurs capacités en résolvant les problèmes d'optimisation multi-objectifs, où le but est de trouver, en une seule exécution de l'algorithme, un certain nombre de solutions proches de l'optimale de Pareto. Beaucoup d'études ont été menées sur différents algorithmes évolutionnaires pour montrer leurs aptitudes à progresser les meilleures solutions, avec une bonne diversité (distribution homogène des solutions sur le front de Pareto) des solutions [65]. Cependant, aucun des algorithmes évolutionnaires multi-objectifs (MOEAs) a une preuve formelle de convergence [65], avec une large diversité, aux véritables solutions en optimales de Pareto. D'après des études qui ont été menées sur plusieurs algorithmes multi-objectifs [65], le NSGA-II [65] est classé parmi les meilleurs et les plus populaires algorithmes jusqu'à ce jour en terme de convergence et de diversité de solutions.

4- Conclusion

Dans ce chapitre, nous avons décrit le fonctionnement et les différents opérateurs d'un algorithme génétique standard, technique d'optimisation mono-objectif.. Nous avons également décrit dans ce chapitre le problème d'optimisation multi-objectifs et quelques algorithmes génétiques de résolution rapportés dans la littérature. Parmi les algorithmes examinés, le NSGA-II semble être aujourd'hui l'une des techniques de référence qui garantit la diversité des solutions Pareto-optimal sans qu'il soit nécessaire de connaître le rayon de niche.

Puisque le problème PSV a plusieurs objectifs à atteindre dans son résolution. Alors, les algorithmes génétiques multiobjectifs seront les plus adéquates dans ce cas.

Parmi ces algorithmes, on a choisi le NSGA-II qui classé parmi les meilleurs algorithmes en terme de convergence et de diversité de solutions.

1- Introduction

L'efficacité des algorithmes génétiques dans la résolution des problèmes qui n'ayant pas des solutions exactes nous a inspiré d'appliquer ce type d'algorithme sur le problème de sélection des vues (PSV). Dans ce chapitre, nous avons d'abord décrit l'implantation du treillis multidimensionnel. Ensuite, nous allons procéder à l'application d'une variante des algorithmes génétiques multi-objectifs (NSGA II) pour sélectionner les vues à matérialiser. Les objectifs du PSV sont la minimisation du cout de requête et de maintenance. Cet algorithme fournit des solutions variées lesquelles offrent différents niveaux de compatibilité entre ces objectifs. Nous avons appliqué cet algorithme et l'algorithme glouton populaire- Benefit per Unit Space (BPUS)- où la performance est reconnue et il est documenté dans [25, 12] sur un exemple typique. Les résultats de cette application nous ont indiqué que les algorithmes génétiques multi-objectifs sont compétitifs par rapport à l'approche glouton.

2- Les préliminaires

2.1- L'entrepôt de données multidimensionnel

Le schéma en étoile d'un entrepôt de données à m-multidimensions se compose d'une table de fait et d'un ensemble de tables de dimension. Une table de fait se compose de m clés de dimensions, dénotées $D_1,D_2,\ldots,D_m$, avec des mesures d'intérêt, dénotées M. Chaque valeur dans la dimension D_i de la table de fait correspond à un seul enregistrement dans la table de dimension correspondante D_i, cette table contienne tous les détails au sujet de cette dimension. Dans une table de dimension, des attributs peuvent être encore organisés dans une structure hiérarchique. Supposons qu'un entrepôt de données multidimensionnel a m dimensions et l'ième dimension à m_i attributs. Il y a $\pi_i^m (2^{m_i} + 1)$ requêtes OLAP possibles (des requêtes SQL group-By), ou des vues.

La Figure 4.1 montre un schéma en étoile pour un entrepôt de données multidimensionnel de trois dimensions STORE, ITEM et DATE et une table de fait.ces tables peuvent être jointes à une grande table représentant un espace multidimensionnel. Le nombre total de requêtes OLAP est alors $(2^4 + 1) \times (2^3 + 1) \times (2^4 + 1) = 2601$

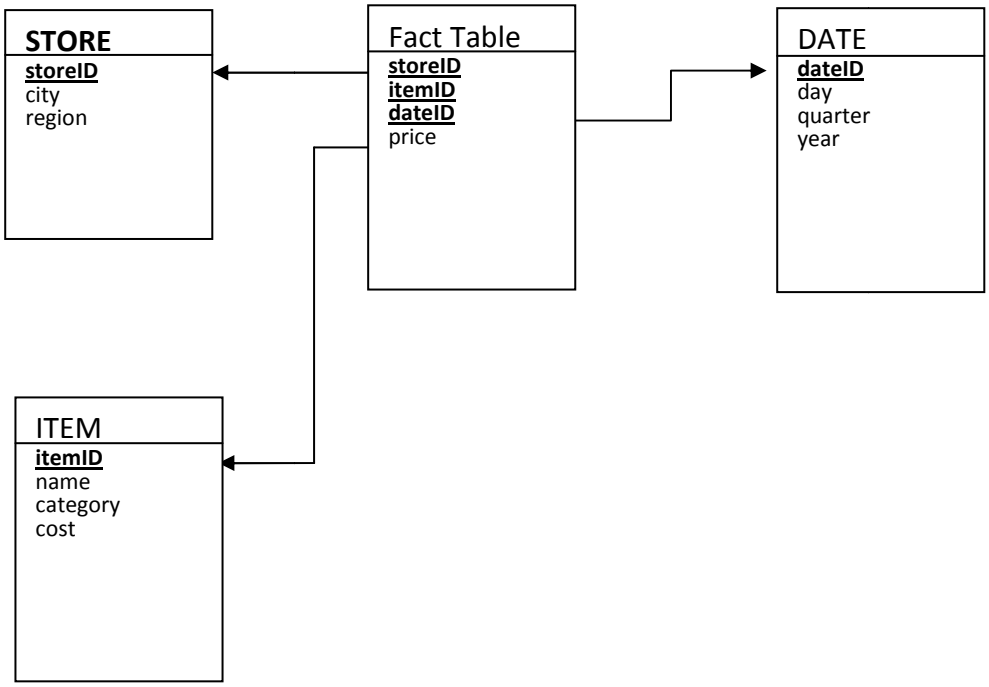


Figure 4.1 : Entrepôt de données multidimensionnel de trois dimensions :Store,Item , et Date

Les requêtes fréquentes de l'entrepôt de données

Les requêtes d'OLAP impliquent habituellement l'agrégation, qui est réalisée par la clause GROUP-BY dans le langage d'interrogation structuré (SQL). Nous supposons que la vue v est la meilleur évaluation de la requête q , si l'ensemble d'attributs apparaissant dans la requête sont les attributs de GROUP-BY de la vue.

2.2- Treillis multidimensionnel

Un treillis se compose des nœuds, représentant les vues possibles qui peuvent être matérialisées, et des arcs représentant des dépendances entre ces vues. Les vues de niveau supérieur représentent la table de fait calculée à partir d'une agrégation des vues dépendantes au niveau inférieur. L'ordre partiel entre des vues est employé pour construire un treillis, où toutes les vues sont dépendantes, directement ou indirectement, au nœud racine du treillis. La vue X serait dépendante de la vue Y, si des requêtes sur la vue X peuvent être répondues en utilisant la vue Y. Les dépendances directes entre des vues représentée par un arc entre les nœuds correspondants, c.-à-d. Y $\preceq$ X (c.-à-d. X est dépend de Y). Tandis que les dépendances indirectes se représentent transitivement, c.-à-d. si le Y $\preceq$ X et le Z $\preceq$ Y, alors cela implique que Z $\preceq$ X. Un nœud dans un treillis est considéré

comme le nœud ancêtre de tous les nœuds qui apparaissent à un niveau inférieur dans le treillis et qu'ils sont dépendent de lui (directement ou indirectement).

2.3- Modèle de coût :

Le problème de trouver l'ensemble optimal des vues à matérialiser (pré_calculer) se réduit alors au problème de trouver l'ensemble optimal de nœuds afin de minimiser la fonction de coût choisie. Pour atteindre cet objectif, on doit définir les différents coûts (requête, la maintenance) associés à l'entrepôt, cela se fait comme suit:

Soit Q = {q_i: i = 1, 2, ..., k} dénote l'ensemble représentatif de requêtes qui pourraient être mieux répondu à partir des vues {u_i: i = 1, 2, ..., k} qui correspond d'une manière exacte dans l'appariement, respectivement. Soit M = {vi: i = 1, 2, ..., m} désigne l'ensemble des vues matérialisées. Soit {F_{UI}: i = 1, 2, ..., k} être la fréquence de demande (interrogation) de la requête q_i et {G_{VI}: i = 1, 2, ..., m} la fréquence de mise à jour de la vue v_i. Le coût de réponse à une requête QC (u, M) est égal au nombre d'enregistrements de la plus petite vue matérialisée ancêtre de u dans le treillis ; cette vue appartienne à l'ensemble M.

Le temps d'exécution global de requête en utilisant *M* sera comme suit [25]:

$$QC_{total} = \sum_{i=1}^{k} f_{u_i}\ QC(u_i, M) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Soit Q_{v1v2} le coût assigné à chaque lien (v_1,v_2) dans le treillis, qui représente le coût de maintien de v_2 en employant des mises à jour de v_1. Le coût de maintenance de la vue v UC (V, M), en présence de l'ensemble des vues matérialisées M égale à la somme des coûts du meilleur chemin d'un ancêtre matérialisé de v.

Le coût total de mise à jour pour un ensemble de vues matérialisées M est [25] :

$$UC_{total} = \sum_{i=1}^{m} g_{v_i}\ UC(v_i, M) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Exemple explicatif :

La figure 4.2 montre un treillis de cube en données pour a une base de données avec de dimensions non-hiérarchique ITEM, STORE, DATE, avec 8 vues (une pour chaque sous-ensemble des dimensions) et leurs tailles associées. Si l'ensemble des vues sélectionnées est M = {SID, SI} (dessinés avec les lignes tirées), la probabilité des requêtes sur toutes les

vues est également probablement 1/8, et 10% de mises à jour à chaque propagation de vue à ses descendants, d'après la formule (1) les coûts sont comme suits :

QC (SID,M) = 1000 ; // QC (SID,M) = la taille de SID parce qu'elle est matérialisée

QC (SI,M)= 800 ; (SI,M) = la taille de SI parce qu'elle est matérialisée

QC (SD,M) = QC (ID,M) = QC (D,M)= QC (SID,M) = 1000= la taille de SID (SID la plus petite vue matérialisée ancêtre de SD, ID, et D dans le treillis)

QC (S,M)= QC (I,M)= QC (none, M)= QC (SI,M)= 800 = la taille de SI (SI la plus petite vue matérialisée ancêtre de S, I, et none dans le treillis)

QC (M) = 4 (1/8) (1000) + 4 (1/8) (800) = 900

Q(M) = 100 + 100 = 200

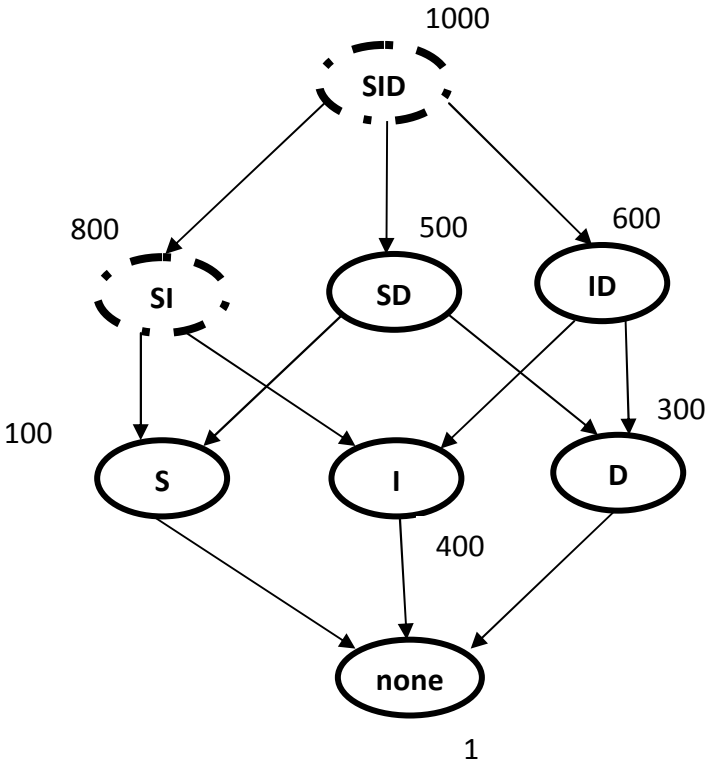


Figure 4.2 : le treillis de l'entrepôt montré dans la figure 4.1

3- Résolution par L'algorithme glouton BPUS

L'abréviation BPUS signifie l'avantage par unité d'espace « benefit per unit space (BPUS) » [25]. À chaque étape, on sélectionne la vue à ajouter à la solution courante. La sélection se base sur la maximisation de l'amélioration de la fonction objective. Par

exemple, quand l'objectif est de minimiser la somme Q (M)+U (M), la vue v ajoutée à la solution courante M à chaque étape est celui pour lequel [45]

$$\frac{(Q(M) + U(M)) - (Q(M \cup \{v\}) + U(M \cup \{v\}))}{r_v} \dots\dots\dots\dots\dots \quad (3) \quad est\ maximal$$

Sachant que :

r_v : Nombre d'enregistrement dans la vue v .

$Q(M) = QC_{total}$ (Cité dans la formule 1)

$U(M) = UC_{total}$ (Cité dans la formule 2)

3.1- Motivation de choix de cet algorithme

La comparaison avec BPUS était afin de fournir un sens plus absolu d'où les solutions des algorithmes génétiques se situent dans l'espace objectif, puisque BPUS garantit une solution à moins de 63% de l'optimal [25].

3.2- Algorithme BPUS

Supposons qu'on a un treillis de cube de données qui possède le nombre d'enregistrements de chaque vue v_i (nœud), la fréquence d'interrogation Fv_i de la requête q_i (répondu par v_i) et la fréquence de mise à jour G_{VI} de la vue v_i.

La vue supérieure doit être toujours inclue dans l'ensemble de vues que nous devrions matérialiser, parce qu'il n'y a aucune autre vue qui ne peut être employée pour répondre à la requête correspondante à cette vue.

Supposons qu'il y a un nombre limité de vues (k). En plus de la vue supérieure, cela que nous pouvons sélectionnées. Après la sélection d'un certain ensemble M de vues, le bénéfique de la vue v relativement à M, dénoté par B (v, M), est défini comme suit.

$$B(v, M) = \frac{(Q(M) + U(M)) - (Q(M \cup \{v\}) + U(M \cup \{v\}))}{r_v} \dots\dots\dots (4)$$

Maintenant, nous pouvons définir l'algorithme glouton pour la sélection d'un ensemble de **k** vues matérialisées.

L'algorithme est montré dans la figure 4.3

```
VM = {vue supérieure };// les vues à matérialisées
VNM= { tous les vues} – {vue supérieure} // les vues non matérialisées
for i=1 to k do
  begin
    // calculer le bénéfique de tous les élément VNM
     For j =1 to (nombre de vue dans VNM) do
          begin
            Calculer B(Vj ,M);
          end;
     Sélectionner la vue Vj qui a le plus grand B(Vj ,M);
     VM = VM ∪ Vj
     VNM = VNM - Vj
 end;
```

Figure 4.3 : l'algorithme glouton

3.3- Les limites de BPUS

Puisque cette approche est exhaustive, donc elle serait l'idéal dans la recherche du front de Pareto optimal (ensemble de solution optimal), mais infaisable au grand espace de recherche de taille2^{2^d} [45].

4- Résolution par les algorithmes génétiques

Comme le problème VSP est NP-difficile [20], on n'espère pas le résoudre de manière exacte lorsque la cardinalité des dimensions augmente. Donc il est indispensable d'utiliser des méthodes métaheuristiques.

Parmi les méthodes métaheuristiques reconnues efficace dans la résolution du problème d'optimisation on a choisi les algorithmes génétiques. Le VSP possédait des objectifs multiples (les coûts de maintenance et de traitement des requêtes), par conséquent l'approche Multi_objective devienne la plus adéquate pour résoudre ce problème.

4.1- Architecture de solution

Pour décrire l'architecture et les composants de notre solution, on a opté le langage de modélisation unifié (UML). Un diagramme de classe UML représentant les composants principaux et leurs rapports est représenté sur la figure 4.4.

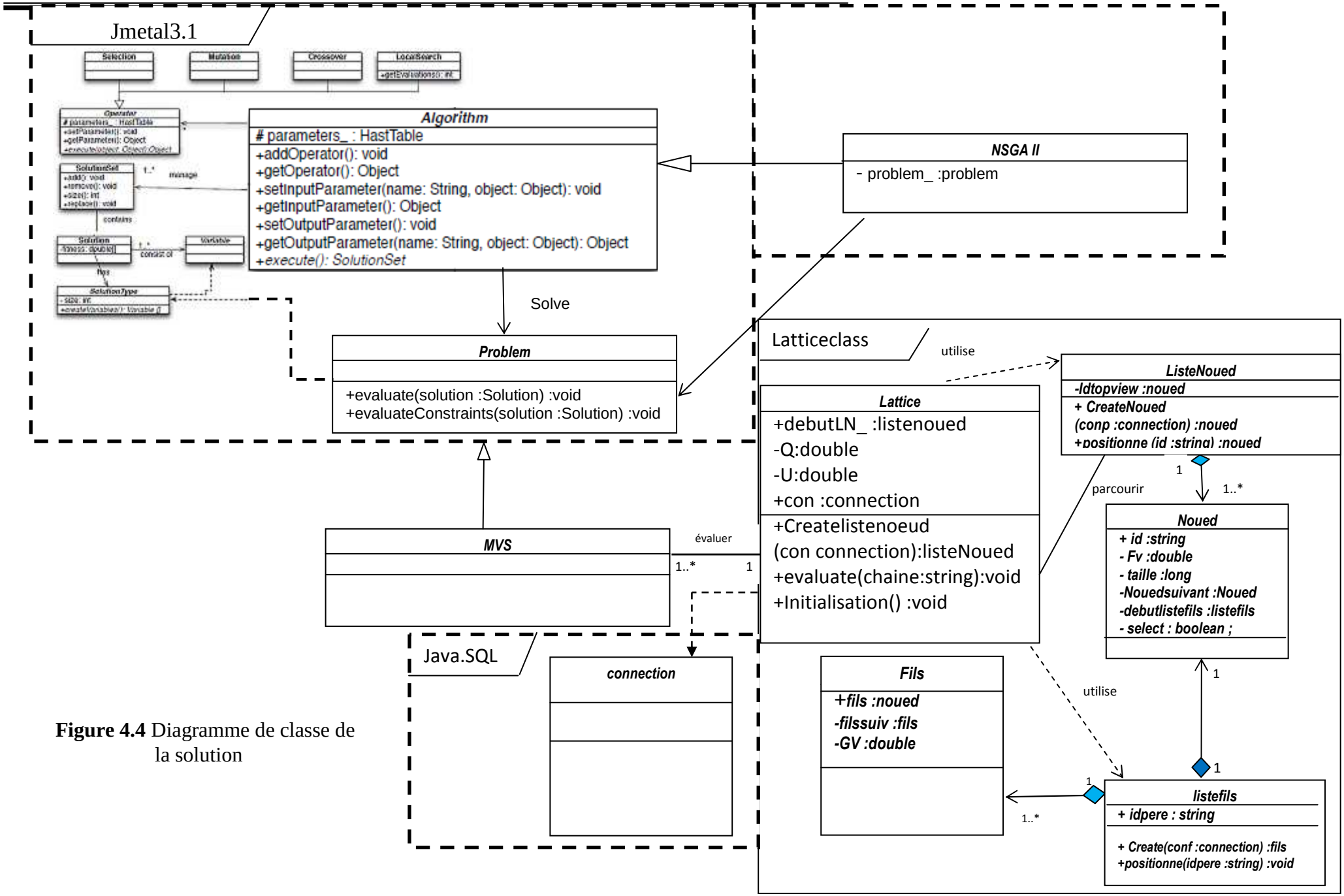


Figure 4.4 Diagramme de classe de la solution

4.2- Description des classes de la solution

Le diagramme se compose de deux groupes de classe (package) l'un est déjà développée et reconnue ; il s'agit jmetal qui est un ensemble riche de classes dont elles peuvent être employées comme blocs constitutifs de metaheuristique à des objectifs multiples. L'autre groupe modélise le treillis (lattice) des vues qui permet d'évaluer les solutions produites par les classes jmetal.

4.2.1- Package Jmetal :

C'est un Framework orienté objet écrit en Java dans le but de développer, d'expérimenter, et d'étudier des algorithmes pour résoudre les problèmes l'optimisation à des objectifs multiples.

L'architecture de base que jMetal compte sur elle un algorithme résout un problème en utilisant un (et probablement plus) SolutionSet - populations -et un ensemble d'objets opérateur [59].

a. Codage des solutions

Parmi les premières décisions qui doivent être prises dans l'utilisation de metaheuristique est de définir comment coder ou représenter les solutions expérimentales du problème à résoudre. La représentation dépend fortement du problème et détermine les opérations (par exemple, recombinaison avec les autres solutions, procédures de recherche locales, etc.) qui peuvent être appliquées. Ainsi, la sélection d'une représentation spécifique a un grand impact sur le comportement du metaheuristique et, par conséquent, dans les résultats obtenus.

Une solution se compose d'ensemble d'objets variables, qui peuvent être de différent types (binaire, real, de nombre entier, etc.) plus un attribut de type tableau pour stocker les valeurs de fitness (adaptation). À l'idée de fournir un arrangement flexible et extensible, chaque solution a associé un type (la classe de SolutionType dans la figure). Le type de solution laisse définir les types de variables de la solution et de les créer, en employant la méthode de createVariables ()[59].

b. Opérateurs

Les techniques de Metaheuristique sont basées sur la modification des solutions existantes ou la production de nouvelles solutions au moyen de l'application de différents opérateurs.

Dans jMetal, n'importe quelle opération qui change ou produise des solutions (ou des ensembles de eux) hérite de la classe opérateur,

Le Framework jMetal incorpore déjà un certain nombre d'opérateurs, qui peuvent être classifiés en quatre classes différentes :

- **Crossover (Croisement)** : Représente les opérateurs de recombinaison ou de croisement utilisés dans les algorithmes évolutionnaires. Parmi les opérateurs inclus sont le croisement binaire (SBX) simulé et le croisement de deux-points pour les codages réels et binaires, respectivement.
- **Mutation** : Représente l'opérateur de mutation utilisé dans les algorithmes évolutionnaires. Les exemples des opérateurs inclus sont la mutation polynôme (codage real) et la mutation bit-flip (codage binaire).
- **Selection** : Ce genre d'opérateur est employé pour exécuter les procédures de sélection dans les EAs. Un exemple d'opérateur de sélection est le tournoi binaire.
- **LocalSearch** : Cette classe est prévue pour représenter des procédures de recherche locales. Elle contient une méthode supplémentaire pour consulter combien d'évaluations ont été exécutées ensuite appliquées.

Chaque opérateur contient les méthodes setParameter () et getParameter (), pour lesquels sont employés pour ajouter et accéder à un paramètre spécifique à l'opérateur. Par exemple, le croisement de SBX exige deux paramètres, une probabilité de croisement (comme la plupart des opérateurs de croisement) plus l'index de distribution (spécifique à l'opérateur), alors qu'un opérateur de mutation d'un paramètre unique qui est la probabilité de mutation[59].

c. Problèmes

Dans jMetal, tous les problèmes héritent de la classe problem. Cette classe contient deux méthodes de base : évaluate () et evaluateConstraints (). Les deux méthodes reçoivent une solution candidate au problème ; le premier l'évalue, et le second détermine la violation globale de contrainte de cette solution. Tous les problèmes doivent définir la méthode evaluate (), alors que seulement les problèmes ayant des contraintes latérales doivent définir evaluateConstraints ()[59].

d. Algorithmes

Algorithm est une classe abstraite qui doit être héritée par la metaheuristique inclus dans le Frameworks. En particulier, la méthode abstraite exécute () doit être mise en application ; cette méthode est prévue pour exécuter l'algorithme, et elle renvoie en conséquence un ensemble de solution SolutionSet (population). Une instance de la classe algorithm exige quelques paramètres spécifiques à l'application, qui peut être ajouté et consulter en employant les méthodes addParameter () et getParameter (), respectivement. De même, un algorithme peut également se servir de quelques opérateurs, ainsi des méthodes pour l'incorporation des opérateurs par addOperator () et pour les obtenir en utilisant getOperator ()[59].

Puisque on a opté l'algorithme NSGA II pour résoudre le problème de sélection des vues matérialisées. Par conséquent, on hérite de l'algorithme la classe NSGA II.

La classe MVS :

L'exploitation de Jmetal dans la résolution des problèmes d'optimisation multiobjective est faite par un simple héritage de la classe problem. Par conséquent, pour modéliser le problème de sélection de vue matérialisée on a hérité la classe MVS (symbolise : materialized view selection). Cette classe évalue les solutions en communiquant la classe lattice qui rend les valeurs des objectives.

4.2.2- Package latticeclass

L'espace de recherche du problème a été représenté par un treillis qui se compose des nœuds et des arcs dirigées et reliées entre eux. Puisque un arc est dirigé donc il peut représente la relation père-fils, pour éclaircir notre cas on a remplacé le terme arc par le terme fils. Après cette description on peut induire deux classes telles que nœud et fils.

Chaque noud possède un nombre de fils variée de zéro à n, ces fils peuvent être organisés sous forme d'une liste, qu'on peut les modéliser par une classe listefils.

La gestion de l'ensemble de nœuds du treillis requis une classe à part, on le renomme par listenoued. La création du treillis et le calcul des coûts de requête et de maintenance (les objectifs du problème) sont encapsulé dans une classe nommé lattice. Dans la suite nous décrirons ces classes .

a. Classe fils : Cette classe modélise la relation de dépendance entre les vues (nœud), elle contient les attributs qui sont :

❖ ***Attributs :***

- **Gv** : contient la valeur du cout de maintien de la vue fils (représenté par l'attribut fils) en employant des mises à jours de la vue père (représenté par l'attribut idpere qui se trouve dans la classe mère du fils –listefils-.
- **Fils** : contient l'objet nœud qui représente l'adresse du nœud fils.
- **Filssuiv** : contient l'adresse du fils suivant.

b. Classe listefils : c'est la classe qui relie le nœud père par les fils, elle comprend un attribut et deux méthodes qui sont :

❖ ***Attributs :***

- **idpere** :qui représente l'identificateur du nœud père
- **conf** : est un attribut de type "connection" .

❖ ***Méthodes :***

- **Create()** :cette méthode permet de créer les fils du nœud père en communiquant l'objet connection qui rend les identificateurs des fils et leurs cout Gv (expliqué au dessus).
- **Positionne ()** : la recherche d'un nœud fils est assurée par cette méthode en donnant l'identificateur du nœud.

c. Classe Noued :cette classe encapsule la vue, elle est géré par les attributs et les méthodes suivants :

❖ ***Attributs :***

- **Id** : identificateur du nœud ;
- **Fv** : représente la fréquence de requête répondu par cette vue
- **taille** : la taille de vue
- **Nouedsuivant** : l'adresse du nœud suivant.
- **debutlistefils** : contient l'objet listefils
- **select** : représente l'état de vue, matérialisée ou non.

❖ ***Méthodes :***

- **Createlistefils** : qui permet de créer le nœud listefils, ce nœud relie le nœud père par les nœuds fils.

d. La classe ListeNoued : c'est la classe qui permet de créer l'ensemble de noued, elle possède un attribut idtopview qui représente l'identificateur du noued père de la vue supérieure (c'est la vue qui doit être inclut dans l'ensemble de vues à matérialiser

parce qu'il n'y a pas de vue qui peut étre utilisé pour répondre à la requête correspondante).

e. La classe lattice *:* c'est la classe principale du package latticeclass. Elle est employée par la classe MVS pour calculer les objectifs du problème en évaluant les solutions resultées de l'algorithme NSGA II.

Cette classe est constitué des attributs et les méthodes suivants :

❖ ***Attributs :***

- **debutLN_** : un attribut qui contient l'adresse de l'objet Listenoued .
- **Q** : représente la valeur du coût global de requête (temps d'exécution) QC_{total} mentionné dans le modèle cité auparavant.
- **U** : représente le cout global de maintenance UC_{total}
- **Conn** : c'est un attribut de type connection, qui permet d'exploiter le méta données de l'entrepôt de données.

  Ce méta données contient des données qui décrit le schéma de l'entrepôt, des statistiques sur la fréquence d'utilisation des requêtes, …etc.

  Pour bien cibler le problème, on n'a pas étudié la manière d'extraire ces données. Nous avons seulement, représenté ces données sous forme d'une base de données à deux relations tels que

  Noued (**idn**,fi,taille,sel,nombrefils) ;

  Fils (**fils,idn**,gi)

❖ ***Méthodes :***

- **Createlistenoued ()** : elle permet de créer listeNoued.
- **Initialisation ()** : elle permet d'initialiser les valeurs des attributs Q, U, et select de la classe nœud. L'initialisation nous permettra de réutiliser ces attributs pour recalculer les valeurs des objectifs dû à la production de nouvelles solutions par l'algorithme NSGA II .
- **Evaluate ()** : Cette méthode permet de calculer les objectifs. Cette tache passe par les étapes suivantes
  - Création d'une liste chainées des vues matérialises en se basant sur la solution résulté de NSGA II .

    La solution se compose d'un ensemble de variables de type binaire.

    Chaque bit dans la chaine binaire représente une vue (nœud).

La valeur de bit signifie que la vue est sélectionnée ou non pour matérialiser.

Exemple : la chaine de 8 bit suivante :10011000 signifie que les vues numéro 1,4,5 sont sélectionnées pour les matérialiser.

- Ordonnancement de cette liste de manière décroissant selon la taille de vue.

Les deux étapes précédentes rendent le calcul du Q facile. Suite au modèle de coût cité au début. Le cout de repense à une requête (vue) égale à la taille de la plus petite vue ancêtre de cette vue, qui nous donne le même résultat si on prend vue par vue de la liste des vues matérialisées ordonnées et on compte le nombre de ces descendants (fils,petit_fils).

Donc Q = Taille de la vue matérialisée * nombre de ces descendants

5- Évaluation expérimentale :

Pour bien exploiter le package Jmetal, on a utilisé le même langage de développement – java - pour implémenter le package classlattice et l'algorithme BPUS.
Les données de la métadonnée employée dans la construction de treillis sont stockées dans une base de donnes réalisée par le SGBD (Système de Gestion de Base de Donnée) postgrés.

5.1- Exemple de problème :

L'algorithme BPUS est infaisable pour un grand espace de recherche. Cette contrainte nous oblige de sélectionner un exemple dont son treillis ne contient pas un grand nombre de nœuds. Parmi les exemples qui satisfont cette contrainte, on a choisi le benchmark TPC-D.
Le benchmark TPC-D modélise un entrepôt d'affaires. Des produits sont achetées de fournisseurs et alors vendu aux clients à un prix de vente. Il y a trois dimensions qui sont : Produit, Fournisseur, et Client. La « mesure » d'intérêt est le total des ventes : ventes (représenté par V). Puisque on a 3 dimensions, donc la table centrale - table de fait- à trois clés étrangères représentant ces dimensions .ainsi, la table de fait peut être représenter par la relation : Vente (**P,C,F**,V)
Les attributs étrangères P,C,F représentent les dimension produit, Client et fournisseur , respectivement. L'attribut V représente les ventes.

Les 3 attributs permettent de créer 8 groupements possibles des attributs. Par conséquent, on obtenu huit requêtes GROUP-BY – qui sont :

-« M » dénote million. La colonne taille contient le nombre d'enregistrement (vue).

ord	Requête	taille	ord	Requête	taille
1	**SELECT** P, SUM(V) **FROM** vente **GROUP** BY P;	**70M**	**5**	**SELECT** P, F, SUM(V) **FROM** vente **GROUP** BY P, F;	**160M**
2	**SELECT** C, SUM(V) **FROM** vente **GROUP** BY C;	**50M**	**6**	**SELECT** F, C, SUM(V) **FROM** vente **GROUP** BY F, C;	**120M**
3	**SELECT** F, SUM(V) **FROM** vente **GROUP** BY F;	**80M**	**7**	**SELECT** P, C, F SUM(V) **FROM** vente **GROUP** BY P, C,F;	**180M**
4	**SELECT** P, C, SUM(V) **FROM** vente **GROUP** BY P, C;	**140 M**	**8**	**SELECT** P, C,F SUM(V) **FROM** vente	**1**

Tableau 4.1 les requêtes GROUP-BY possibles sur le benchmark TPC-D

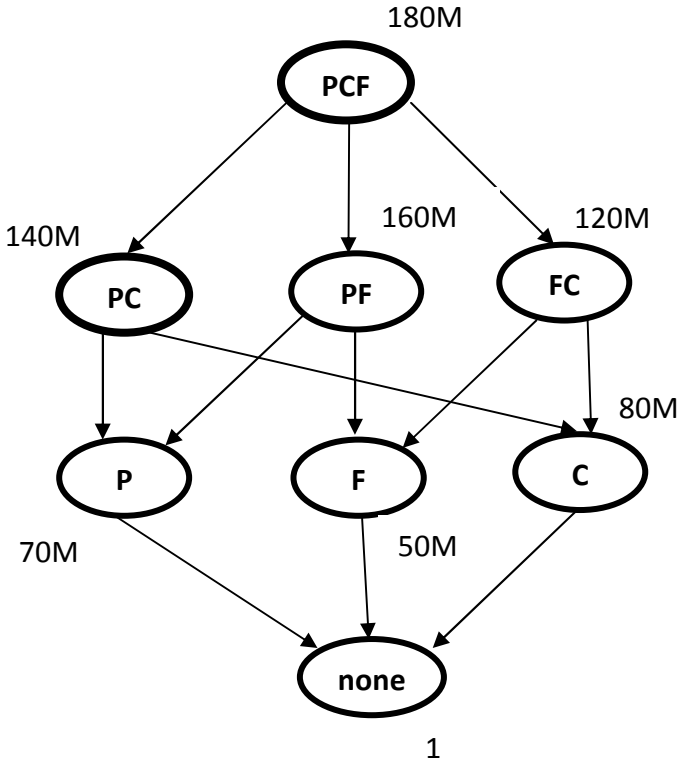


Figure 4.5 le treillis du benchmark TPC-D

La Figure 4.4 illustre le treillis de dépendance du benchmark TPC-D, Chaque nœud du treillis représente une requête OLAP ou une vue. L'arc qui part du nœud **PC** au nœud **C** représente le fait que la requête OLAP **C** peut être traitée par la requête OLAP **PC**.

5.2- Résolution par NSGA II :

Le code suivant montre la méthode « Constructeur » du classe MVS, cette méthode utilise dans la création de l'objet MVS par la classe NSGA II par l'instruction suivante :

problem = new MVS("Binary", 8);

Le premier paramètre sert à représenter le type de codage, le second pour le nombre de variable. Dans notre cas 8 variables nécessaire, chacune représente une vue.

La vue	PCF	C	P	PC	PF	None	F	FC
Chaine binaire	bit1	bit2	bit3	bit4	bit5	bit6	bit7	bit8

```
1  public MVS(String solutionType, Integer numberOfVariables) throws ClassNotFoundException {
2  numberOfVariables_   = numberOfVariables.intValue() ;
3   numberOfObjectives_  = 2                    ;
4  numberOfConstraints_ = 0                    ;
5   problemName_        = "MVS"                ;
6   solutionType_ = new BinarySolutionType(this) ;
7  LC = new LatticeClass();
8 }
```

Paramètres de l'algorithme:

Le paramétrage nécessaire de NSGA II est donné dans le tableau suivant :

Paramètre		Valeur	Explication
Populationsize		8	La taille de population généralement égale au nombre de variable
maxEvaluation		64	maxEvaluation = Nombre de génération * la taille de population
Crossover	Type	"SinglePointCrossover"	Croisement à une seule

			position
	Probability	0.9	Probabilité de croisement
Mutation	Type	"BitFlipMutation"	
	Probability	1/8	
Selection		"BinaryTournament2"	

Tableau 4.2 Paramètres de l'algorithme NSGA-II appliqué à la sélection des vues

Les autres paramètres tel que :

Les fréquences de requête et de maintenance sont supposées égales est de valeur 1/8.

Aussi pour simplifier le calcul du coût de maintenance des vues, on a supposé que le coût de maintenir une vue v1 en utilisant des mises à jour d'un de ses ancêtres v2 est égal à 10% de la taille de v2 [58] .

5.3- Les résultats

Les sorties ou résultats du framework Jmetal sont écrits dans les deux fichiers -format texte- qui portent les noms suivants VAR et FUN.

Fichier VAR : ce fichier contient les valeurs des objectifs de la dernière génération- 512 ème evaluation (c-a-d 64 générations). Ces valeurs représentent le front de Pareto. Puisque dans notre cas le paramètre- taille population- égal à 8, donc ce fichier a 8 lignes. Chaque ligne représente un individu de la population.

On a remarqué que les lignes sont identiques, cela signifie que l'algorithme nous a donné qu'une seule solution.

La première valeur représente le temps d'exécution global de requête qui est symbolisé par Q dans le modèle de coût.

La seconde représente le coût de maintenance de l'ensemble M symbolisé par U.

L'ensemble M contient les vues à matérialiser.

Fichier FUN : ce fichier contient les valeurs des solutions, qui sont aussi identique.

Nous avons constaté que les résultats obtenus par les deux algorithmes sont semblable.

Par conséquent, les performances de l'algorithme PBUS se reflète sur les algorithmes génétiques multi-objectifs et surtout la variante NSGA II.

6- Conclusion

Nous avons appliqué dans ce chapitre les algorithmes génétiques multi-objectifs dans le problème de sélection des vues à matérialiser, la variante adoptée dans notre étude est la NSGA II. Au début, on a présenté le modèle de coût qui permet d'évaluer les solutions données par l'algorithme NSGA II. Ensuite, on a déployé la solution en utilisant le langage de modélisation UML. Cette solution se compose de deux composants.

Le composant principal a été déjà développé, il s'agit le framwork Jmetal,qui est un ensemble de classes encapsulant les algorithmes génétiques multi-objectifs.

Le second composant a été développé dans le but de créer et d'explorer un treillis de vues en évaluant les solutions données par le premier composant.

Enfin, pour valider cette solution, on a choisi un exemple de petite cardinalité de dimensions abordable par les algorithmes génétiques et par l'algorithme glouton Benefit per Unit Space (BPUS).

Bibliographie

[01] A. Shukla, P. Deshpande, and J. F. Naughton. **Materialized view selection for multidimensional datasets**. Proceedings of the International Conference on Very Large Databases,pages 488-499. August 1998.

[02] B. G. Lindsay, L. M. Haas, C. Mohan, H. Pirahesh, and P. F. Wilms. **A snapshot differential refresh algorithm**. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 53–60. June 1986.

[03] E. Benitez, C. Collet, and M. Adiba. **Entrepôts de données : caractéristiques et problématique**. Revue TSI, 20(2). 2001.

[04] Chaudhuri S., Dayal U.**An Overview of Data Warehousing and OLAP Technology**, ACM SIGMOD Record, 26(1). 1997.

[05] Chirkova, R. & Li, C. & Li, J. **Answering queries using materialized views with minimum size**. The International Journal on Very Large Data Bases(VLDB J), pages 191-210. 2006.

[06] Codd E.F.,**Providing OLAP (on-line analytical processing) to user-analysts : an IT mandate**. Technical Report, E.F. Codd and Associates. 1993.

[07] Cordon, O., F. Herrera et P. Villar. **Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base**. IEEE Transactions on Fuzzy Systems 9(4),pages 667–674. 2001.

[08] Corne, D., J.D. Knowles et M.J. Oates. **The pareto envelope-based selection algorithm for multi-objective optimization**. In: Proceedings of the sixth International Conference on Parallel Problem Solving from Nature, pages 839–848. 2000.

[09] Deb, K., A. Pratap, S. Agarwal et T. Meyarivan . **A fast and elitist multiobjective genetic algorithm: Nsga-ii**. IEEE Transactions on Evolutionary Computation 6(2), pages 182–197. 2002.

[10] Dhote, C.A and M.S Al. **Materialized view selection in data warehousing**. Fourth International Conference on Information Technology, ITNG 2007, April 2-4, IEEE Computer Society Washington, DC, USA.,Pages 843-847. 2007.

[11] B. Dinter, C. Sapia, M. Blaschka, and G. Höfling. **OLAP Market and Research : Initiating the Cooperation**. Computer Science and Information Management, 2(3). 1999.

[12] H. Gupta and I. S. Mumick. **Selection of views to materialize in a data warehouse**. IEEE Trans. on Knowledge and Data Eng., 17(1):pages 24–43. January 2005.

[13] E. Baralis, S. Paraboschi, and E Teniente. **Materialized view selection in a multidimensional database**.Proceedings of the International Conference on Very Large Databases, pages 156–165 . August 1997.

[14] Fonseca, C.M. et P.J. **Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization**. In: Proceedings of the Fifth International Conference on Genetic Algorithms (S. Forrest, Ed.). Morgan Kaufmann. San Mateo,California. pages 416–423. 1993.

[15] Goldberg, D.E. et J. Richardson. **Genetic algorithms with sharing for multimodal function optimization**. In: Second International Conference on Genetic Algorithms and their application. Cambridge, Massachusetts, United States, Pages 41–49. 1987.

[16] Goldberg, D.E. **Algorithmes génétiques. Exploration, optimisation et apprentissage automatique**. Addison-Wesley. France,1994.

[17] Golfarelli M., Maio D., Rizzi S., **Conceptual design of Data Warehouse from E/R Schemas**, Proceeding of the 31st Hawaii International Conferene on System Sciences, Kona (Hawaii, USA). 1998.

[18] Gray J., Bosworth A., Layman A., Pirahesh H., **Datacube: A Relational Operator Generalizing Group-By, Cross-Tab and Sub-Total** , Proceeding of the 12 International Conference on Data Engineering, pages 152-159. 1996.

[19] Lee, M., & Hammer, J**. Speeding Up Materialized View Selection in Data Warehouses Using a Randomized Algorithm**. International Journal of Cooperative Information Systems (IJCIS),10(3), pages 327-353. 2001.

[20] Gupta, H. **Selection of Views to Materialize in a Data Warehouse**. International Conference on Database Theory (ICDT). Delphi, Greece, pages 98-112. 1997.

Baralis, E., Paraboschi, S., & Teniente,E. **Materialized Views Selection in a Multidimensional Database**. International Conference on Very Large Data Bases. Athens, Greece, pages 156-165. 1997.

Yang, J., Karlapalem, K., & Li, Q. **Algorithms for Materialized View Design in Data Warehousing Environment.** International Conference on Very Large Data Bases. Athens, Greece, pages 136-145,1997.

[21] H. Gupta. **Selection and maintenance of views in a data warehouse**. Ph.d. thesis, Stanford University, September 1999.

[22] J. Yang, K. Karlapalem, and Q. Li. **Algorithms for materialized view design in data warehousing environment**. Proceedings of the International Conference on Very Large Databases, pages 136–145. August 1997.

[23] H. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. **What can hierarchies do for your data warehouses**.Proceedings of the International Conference on Very Large Databases, pages 530–541, September 1999.

[24] Golfarelli, M., & Rizzi, S.**View materialization for nested GPSJ queries.** International Workshop on Design and Management of Data Warehouses (DMDW).Stockholm, Sweden, pages 1-10. 2000.

[25] Harinarayan, V., Rajaraman, A., & Ullman, J.**Implementing Data Cubes Efficiently.** ACM SIGMOD International Conference on Management of Data (SIGMOD). Montreal, Canada,pages 205-216. 1996.

Shukla, A., Deshpande, P., & Naughton, J. **Materialized View Selection for Multidimensional Datasets**. International Conference on Very Large Data Bases (VLDB). New York City, USA,pages 488-499. 1998.

Gupta, H., & Mumick, I.S. **Selection of Views to Materialize Under a Maintenance Cost Constraint**. International Conference on Database Theory (ICDT). Jerusalem, Israel, pages 453-470. 1999.

[26] Holland, J**. Adaptation in natural and artificial and systems**. 2nd edition. IT Press, 1992.

[27] Horn, J. et N. Napfliotis. Multiobjective **optimization using the niched pareto genetic algorithm**. In: Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence. Vol. 1. pages 82–87, 1994.

[28] Hwang, H.S. **Automatic design of fuzzy rule base for modelling and control using evolutionary programming**. IEE Proceedings on Control Theory and Applications pp. 9–16, 1999.

[29] William Inmon. **Building the Data Warehouse**. QED Technical Publishing Group, Wellesley, Massachusetts, U.S.A., 1992, 1992.

[30] Inmon W.H.. **Building the Data Warehouse**, John Wiley&Sons, ISBN 0471-14161-5, 1994.

[31] J. A. Blakeley, P. Larson, and F. W. Tompa. **Efficiently updating materialized views**. Proceedings of the ACM SIGMOD International Conference on Management of Data, pp 61–71, June 1986.

[32] J. Ullman. **Efficient implementation of data cubes via materialized views**. in the Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96), pp 386–388, 1996.

[33] Kalnis, P., Mamoulis, N., & Papadias, D.**View selection using randomized search**. Data & Knowledge Engineering, 42(1), 89-111, 2002.

[34] Kimball R., **The data warehouse toolkit**, John Wiley and Sons, 1996.

[35] L.Bellatreche and Boukhalfa K. **An evolutionary approach to schema partitioning selection in a data warehouse environment**. Proceeding of the International Conference on Data Warehousing and Know-ledge Discovery (DAWAK'2005), pp 115–125, August 2005.

[36] Lee, M., & Hammer, J. **Speeding Up Materialized View Selection in Data Warehouses Using a Randomized Algorithm**. International Journal of Cooperative Information Systems (IJCIS),10(3), 327-353,2001.

[37] M. T. ¨ Ozsu and P. Valduriez. Principles **of Distributed Database Systems** : Second Edition. PrenticeHall, 1999.

[38] Man, K.F., K.S. Tang et S. Kwong. **Genetic algorithms: Concepts and applications**.IEEE Transaction on industrial electronics 43,pp 519–534, 1996.

[39] P. Marcel. **Manipulations de Données Multidimensionnelles et Langages de Règles**. PhD thesis, Institut National des Sciences Appliques de Lyon, France, 1998.

[40] Mendelzon A.O., Vaisman A.A., **Temporal Queries in OLAP**,Proceedings of 26th International Conference on Very Large Data Bases - VLDB 2000, Cairo (Egypt), September 10-14, 2000.

[41] Michalewicz, Z. **Genetic Algorithms + Data Structures = Evolution Programs**. Springer-Verlag, 1992.

[42] Mistry H., Roy P., Sudarshan S. & Ramamritham K. (2001). **Materialized View Selection and Maintenance Using Multi-Query Optimization**. ACM SIGMOD International Conference on Management of Data (SIGMOD). Santa Barbara, California, 307-318, 2001.

[43] The OLAP Report - **http ://www.olapreport.com/fasmi.htm**, 2004.

[44] Pedersen T.B., Jensen C.S., **Research Issues in Clinical Data Warehousing**, SSDBM'98, Capri (Italy), July 1998,.

[45] Lawrence, M . **Multiobjective Genetic Algorithms for Materialized View Selection in OLAP Data Warehouses.** ACM GECCO 2006,Us , pp 699-706, 2006.

[46] Red Brick Systems. **Star schema processing for complex queries**. White Paper, July 1997.

[47] Ross, K., Srivastava, D., & Sudarshan, S. **Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time**. ACM SIGMOD International Conference on Management of Data (SIGMOD). Montreal, Canada, pp 447-458, 1996.

[48] Lee, M. and J.Hammer. speeding **up warehouse physical design using a randomized algorithm**. Int J. Cooperative inform. Syst, 10: 327-353, 2001.

[49] S. Chaudhuri and V. Narasayya. **Autoadmin 'what-if' index analysis utility**. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 367–378, June 1998.

[50] Yu, J.X,X. Yao, C.H Choi and G. Gou. **Materialized view selection as constrained evolutionary optimization**. IEEE Trans.Syst.Man Cybernetics part C, 33:pp 458-467, 2003.

[51] Schaffer, J.D. **Multiple objective optimization with vector evaluated genetic algorithms**. In: Proceedings of the 1st International Conference on Genetic Algorithms.pp. 93–100, 1985.

[52] Srinivas, V. et K. Deb.**Muiltiobjective optimization using nondominated sorting in genetic algorithms**. Evolutionary Computation 2(3), 221–248, 1993.

[53] V. Harinarayan, A. Rajaraman, and J. Ullman**.Implementing data cubes effeciently**. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 205–216, June 1996.

[54] Panos Vassiliadis and Timos K. Sellis. **A Survey of Logical Models for OLAP Databases**. SIGMOD Record, 28(4) :pp 64–69, 1999.

[55] M-C Wu and A.P. Buchmann. **Research Issues in Data Warehousing In BTW** German Database Conference, 1997.

[56] Wu, C.J. et G.Y. Lin. **Design of fuzzy logic controllers using genetic algorithms**. In: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics Vol. 6. SAAA, pp. 104–109, 1999.

[57] Zitzler, E. et L. Thiele. **Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach**. IEEE Transactions on Evolutionary Computation 3(4), pp 257–271,1999.

[58] I.S. Mumick, D. Quass, B.S. **Mumick, Maintenance of data cubes and summary tables in a warehouse**, in:Proc. ACM SIGMOD, 1997.

[59] J.Antonio J. Nebro, J.Juan. Durillo, **jMetal 3.1 User Manual**, pp 12-16, 2010.

[60] Ghali K. **Méthodologie de conception système a base de plateformes reconfigurables et programmables**, Mars 2005.

[61] O.Guenounou. **Méthodologie de conception de contrôleurs intelligents par l'approche génétique- application à un bioprocédé**, Thèse doctorale, l'Université Toulouse III - Paul Sabatier, Avril 2009.

[62] C.A Dhote,M.S Ali,**Materialized view selection in Data warehousing: A survey**,Journal of Applied sciences 9 (3),2009.

[63] Serna Encinas M.T**, Entrepôts de données pour l'aide à la décision médicale : conception et expérimentation**, Thèse doctorale, Université Joseph FOURIER, Juin 2005

[64] J. Yang, K. Karlapalem, and Q. Li. **Algorithms for materialized view design in data warehousing environment.** Proceedings of the International Conference on Very Large Databases, pages 136–145, August 1997.

[65] Lee, M. and J.Hammer,. **speeding up warehouse physical design using a randomized algorithm**. Int J. Cooperative inform. Syst, 10: 327-353, 2001.

[66]Yu, J.X,X. Yao, C.H Choi and G. Gou. **Materialized view selection as constrained evolutionary optimization**. IEEE Trans.Syst.Man Cybernetics part C, 33:458-467, 2003.

[67] J-M Franco. Le Data Warehouse (Le Data Mining). Editions Eyrolles, Paris, 1997.
[68] A. Doucet and S. Gangarski. **Entrepôts de données et Bases de Données Multidimensionnelles**, Chapitre 12 du livre : Bases de Données et Internet, Modèles, langages et systèmes. Editions Hermès, 2001.

[69] D. Theodoratos,W. Xu,A.Simitsis.**Materialized View Selection for Data Warehouse Design**. Encyclopedia of Data Warehousing and Mining,Second Edition,pp 1183-1185, 2009.

[70] L. Bellatreche**, Utilisation des vues matérialisées, des index et de la fragmentation dans la conception logique et physique d'un entrepôt de données**.
Thèse doctorale. Université Clermont-Ferrand II, France, 2000.

Conclusion et perspectives

a. Conclusion

La croissance contenue de la taille des entrepôts de données rend le champ de recherche des techniques d'optimisation des requêtes très délicat. Parmi ces techniques, on a intéressé par les vues matérialisées. La problématique suggérée dans ce domaine est qu'elles sont les vues adéquates pour la matérialisation.

Nous avons présenté dans ce document une méthode d'exploration de l'espace de solutions représenté par un treillis pour mieux choisir les vues à matérialiser qui permet d'accélérer l'exécution des requête au futur en tenant compte à la fiabilité de ses réponses.
Cette méthodologie d'exploration et d'optimisation est basée sur des algorithmes génétiques multiobjectifs.

Nous avons dans ce document répondu à la problématique présentée en introduction, à savoir s'il était possible de définir une méthode d'exploration de l'ensemble représentant les vues candidates sans passer par des simulations. La méthode que nous proposons s'appuie essentiellement sur l'évaluation de performances à travers un modèle de coût.

En effet, ce choix d'optimisation multiobjectifs est essentiel afin que la sélection des vues à matérialiser soit efficace tant du point de vue du coût d'exécution des requêtes que de la maintenance de ces vue. Cette analyse a été rendue possible grâce à des études que nous avons menées sur les algorithmes génétiques d'optimisation multiobjectifs et leurs adaptations dans le domaine - sélection de vues-.

Nous avons montré dans ce document que cette méthode donne des résultats semblables à l'algorithme BPUS qui est reconnu efficace dans un espace de recherche restreint.

Le résultat fourni par cette méthodologie est un ensemble de fronts de Pareto qui nous permet de choisir la bonne configuration de l'entrepôt de données.

Aujourd'hui, pour sélectionner des vues dans un entrepôt qui a plus de dimensions dans son schéma constitue une entrave à son administrateur.
Alors, les experts ont besoin de ces explorations automatiques pour parvenir à un bon choix de paramétrage de leurs applications. Comme nous l'avons montré dans ce travail, les algorithmes génétiques multiobjectifs sont bien adaptés à ce type d'exploration automatique.

b. Perspectives

Nous avons développé une méthodologie statique pour sélectionner les vues adéquates à matérialiser. Dans ce travail nous n'avons pas abordé la nature dynamique de l'analyse - aide à la décision-. Surtout, pour les requêtes ad hoc qui sont formulées par les experts afin de trouver des tendances intéressantes, sont souvent difficiles à prévoir. Par conséquent, les résultats de la méthode de sélection statique peuvent être bientôt mis à jour. Cela signifie que les administrateurs doivent exécuter régulièrement l'algorithme de vision statique de sélection pour ajuster l'ensemble des vues matérialisées et répondre aux besoins des utilisateurs.

Ce travail est basé sur une technique unique d'optimisation qui n'est pas corrélée avec les caractéristiques de complexité du problème. Intuitivement, il serait souhaitable d'utiliser des techniques hybrides permettant une meilleure adéquation au problème à traiter soit a priori de manière statique ou d'une manière dynamique et adaptative.

Comme perspective à cette méthodologie d'exploration, nous citons une technique hybride qui se base sur les algorithmes génétiques et la colonie de fourmi, en divisant l'ensemble des vues à deux catégories, statique et dynamique. Cette combinaison permet de bénéficier d'une part, le temps de réponse amélioré par l'approche statique, et d'une autre part la possibilité de l'auto-administration l'approche dynamique.

De nombreuses autres problématiques passionnantes existent qui pourraient faire émerger de nombreux thèmes de recherche.

Résumé : Le travail présenté dans ce mémoire concerne l'optimisation des requêtes dans l'entrepôt de données (Data Werehouse), plus particulièrement la technique des vues matérialisées. Un important problème d'optimisation a été suggéré dans cette technique, il s'agit du problème de sélection de vues. Il consiste a sélectionné l'ensemble des vues à matérialiser pour accélérer dans le futur l'exécution des requêtes. Des contraintes ont été exigées dans le processus de sélection, qui sont orientées au système et à l'utilisateur. La matérialisation de tous ces vues accélère l'exécution de n'importe quelle requête mais nécessite un coût considérable pour les maintenir, Sachant que la maintenance des vues rend les résultats des requêtes plus bénéfiques pour les décideurs. Alors, le problème devient une optimisation multi-objectifs. Ce mémoire propose une solution à ce problème en plusieurs étapes : dans la première étape, une formulation mathématique du problème a été développée. Dans la seconde étape, on a construit un treillis des vues possibles. Dans la troisième étape on a appliqué les algorithmes génétiques multi-objectifs sur le treillis développé avant. La variante adoptée dans la résolution est NSGAII (Nondominated Sorting Genetic Algorithm).

Mots Clés : Vue matérialisée, Entrepôt de données, les algorithmes génétiques multi-objectifs, Optimisation, requête OLAP

Abstract: The work presented in this memory is related to the optimization of the query in the data warehouse, more particularly the technique of the materialized views. Big problems of optimization were suggested in this technique, which deals with the problem of selection of views. The technique consists of selecting the whole of the views to be materialized to accelerate in the future the execution of the query. Constraints were required in the selection process, which is directed with the system and the user. The matérialisation of all the views accelerates the execution of any query but requires a considerable cost to maintain them, knowing that the maintenance of the views returns the results of the query more beneficial for the decision makers.Then, the problem becomes an optimization multi-objectives. This memory proposes a solution with this problem in three steps: In the first step, we used and developed a mathematical formulation.In the second step, we built a lattice of the possible views.In the third step, we applied the multi-objectives genetic algorithms on the lattice so far developed . The variante adopted in the resolution is NSGA II (Nondominated Sorting Genetic Algorithm).

Keywords: Materialized view,Data warehouse,multi-objectives genetic algorithm,Optimisation, OLAP Query.

ملخص : يتعلق العمل المقدم في هذه الأطروحة بتعظيم الاستفادة من الاستعلامات في مخزن المعلومات، وخصوصا تقنية الأجوبة المخزنة .ظهر مشكل مهم لتحسين هذه التقنية، يتمثل في كيفية اختيار هذه المجموعة من الاجوبة المخزنة.يتم اختيار هذه المجموعة لتحقيق هدف تسريع الاستعلامات في المستقبل. في عملية الاختيار يجب مراعاة شروط معينة ، تتعلق بالنظام والمستخدم. اختيار كل الأجوبة للتخزين يعجل في الاجابة على اي طلب ، بالمقابل تحتاج الى تكلفة كبيرة لتحديثها، مع العلم أن تحديثها يجعل نتائج الاستعلام أكثر فائدة بالنسبة لصانعي القرار .وبالتالي يصبح مشكل التحسين متعدد الأهدف .هذه الرسالة اقترحت حلا لهذه المشكلة في عدة خطوات: في الخطوة الأولى، وضع الصيغة الرياضية للمشكلة .ثم في الخطوة الثانية، انشأت شبكة للاجوبة الممكنة و المحتملة. في الخطوة الثالثة قمنا بتطبيق الخوارزميات الوراثية متعددة الاهدف على شبكة التي صممت من قبل .الخوارزميات الذي اعتمدناه هي NSGAII (Nondominated Sorting Genetic Algorithm)

.

الكلمات المفتاحية : الأجوبة المخزنة، مخزن المعطيات، الخوارزميات الوراثية متعددة الاهدف ،التحسينات، استعلامات OLAP