La fragmentation dans les entrepôts de données : une approche basée sur les algorithmes génétiques

Ladjel Bellatreche*, Boukhalfa Kamel **

* LISI/ENSMA
Futuroscope - France
bellatre@ensma.fr
** Université de Laghouat Route de Ghardaia,BP 37G
Laghouat Algérie
k.boukhalfa@mail.lagh-univ.dz

Résumé. La fragmentation horizontale est une technique d'optimisation non redondante de requêtes décisionnelles de type ROLAP. L'utilisation de cette technique dans les entrepôts de données représente un enjeu plus important que dans un contexte de bases de données traditionnelles. Cette importance est due au différents choix des tables (de dimensions ou des faits) à fragmenter. Dans le contexte des entrepôts, la fragmentation n'a un sens que si la table des faits est partitionnée en fonction des schémas de fragmentation des tables de dimensions. Mais ce type de fragmentation de la table des faits pourrait engendrer un nombre important de fragments qui rendrait le processus de maintenance très coûteux. Afin de réduire ce nombre ou le rendre contrôlable par l'administrateur de l'entrepôt, nous proposons l'utilisation d'un algorithme génétique. Ce dernier a pour but de sélectionner les tables de dimension à fragmenter pour (1) éviter l'explosion du nombre de fragments de la table des faits et (2) garantir une meilleure performance d'exécution des requêtes. Notre algorithme génétique est développé sous visual C et validé par une étude expérimentale en utilisant le banc d'essai APB-1 release II.

1 Introduction

Les principales caractéristiques des entrepôts de données sont leur grande taille et la complexité des requêtes décisionnelles dues aux opérations de jointure et d'agrégation. Plusieurs techniques d'optimisation ont été proposées pour réduire le coût d'exécution de requêtes. Ces techniques peuvent être divisées en deux catégories : *structures redondantes* et *structures non redondantes*. Les structures redondantes nécessitent un espace de stockage et un coût de mises à jour. Parmi ces dernières, on peut citer les vues matérialisées et les index [Kotidis et Roussopoulos, 1999, Gupta, 1999]. La deuxième catégorie ne nécessite pas de coût de stockage. C'est le cas des techniques de fragmentation (horizontale et verticale) [Sanjay *et al.*, 2004]. La plupart des systèmes commerciaux comme ORACLE, DB2 proposent les deux structures [Zilio *et al.*, 2004, Sanjay *et al.*, 2004]. Dans ce papier nous nous intéressons à une structure non redondante, à savoir la fragmentation horizontale. Plusieurs travaux de recherche et des produits commerciaux ont montré son utilité dans le processus d'optimisation des requêtes

décisionnelles [Sanjay *et al.*, 2004, Bellatreche *et al.*, 2004].

La fragmentation horizontale dans les entrepôts de données représente un enjeu plus important que dans un contexte de base de données relationnelles ou objet. Cette importance est due au choix des tables (de dimensions ou des faits) à fragmenter. Les choix possibles sont les suivants :

1. Fragmenter seulement quelques tables de dimensions. Ce choix n'est pas souhaitable pour les requêtes décisionnelles, pour deux raisons : (i) les tailles des tables de dimensions sont généralement petites, (ii) les requêtes décisionnelles accèdent à la table des faits qui est très volumineuse. En conséquence, toute fragmentation ne prenant pas en considération la table des faits est à exclure.
2. Partitionner seulement la table des faits. La table des faits est composée des clés étrangères des tables de dimensions et des données brutes. Ces dernières représentent des mesures numériques, comme le montant des ventes, le nombre d'articles soldés, etc. Généralement, dans une requête décisionnelle, nous trouvons rarement des prédicats de sélection définis sur la table des faits. De plus, une requête décisionnelle typique commence par la sélection des critères selon lesquels s'effectuera l'analyse sur les tables de dimensions, puis s'orientera sur la valeur des indicateurs pour la sélection effectuée [Noaman et Barker, 1999]. Ce choix est donc moins souhaitable pour le processus de la fragmentation.
3. Fragmenter totalement ou partiellement les tables de dimensions et utiliser leurs schémas de fragmentation pour partitionner la table des faits (dans ce cas, la table des faits est alors partitionnée en utilisant la fragmentation dérivée). Ce choix est bien adapté aux entrepôts, car il prend en considération les exigences des requêtes décisionnelles, ainsi que les relations entre les tables de dimensions et la table des faits [Bellatreche *et al.*, 2004].

   Supposons que chaque table de dimension D_i fragmentée a m_i fragments : $\{D_{i1}, D_{i2}, ..., D_{im_i}\}$, où chaque fragment D_{ij} est défini par :
   $D_{ij} = \sigma_{cl_j^i}(D_i)$ avec cl_j^i $(1 \leq i \leq g, 1 \leq j \leq m_i)$ représentant une conjonction de prédicats simples.

   Ainsi le schéma de fragmentation de la table des faits F est défini comme suit : $F_i = F \ltimes D_{1i} \ltimes D_{2i} \ltimes ... \ltimes D_{gi}$, avec $\ltimes$ qui représente l'opération de la semi-jointure.

Exemple 1 *Considérons un schéma en étoile avec trois tables de dimension (Client, Temps, Produit) et une table des faits Ventes. Supposons que seule la table de dimensions CLIENT soit décomposée en deux fragments horizontaux Client_1 et Client_2 définis par les clauses suivantes :*

$$Client\_1 = \sigma_{Sexe='M'}(CLIENT) \quad (1)$$

$$Client\_2 = \sigma_{Sexe='F'}(CLIENT) \quad (2)$$

La table des faits VENTES peut alors être décomposée en utilisant la fragmentation horizontale dérivée, en deux fragments horizontaux Ventes_1 et Ventes_2 tels que :

$$Ventes\_1 = VENTES \ltimes Client\_1 \quad (3)$$

$$Ventes\_2 = VENTES \ltimes Client\_2 \quad (4)$$

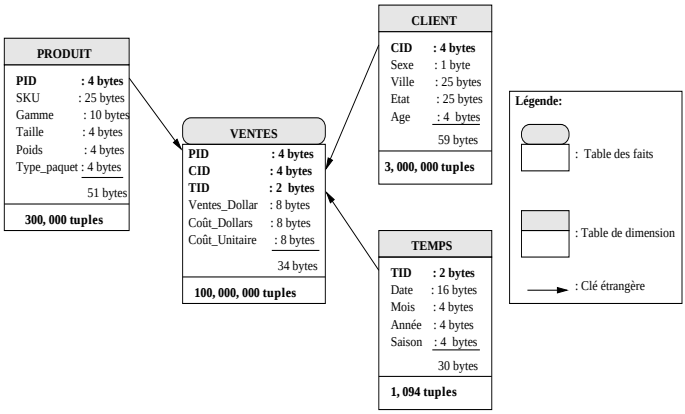

FIG. 1 – Schéma en étoile des activités de ventes

Le schéma en étoile initial (Ventes, Client, Produit, Temps) est la juxtaposition de deux sous-schémas en étoile S_1 et S_2 tels que : S_1 : (Ventes_1, Client_1, Produit, Temps) (ventes des clients masculins) et S_2 : (Ventes_2, Client_2, Produit, Temps) (ventes des clients féminins).

2 Complexité de la fragmentation dérivée de la table des faits

Soit un schéma en étoile de d tables de dimensions et une table des faits. Soit g ($g \leq d$) le nombre de tables de dimensions horizontalement fragmentées. Le nombre de fragments horizontaux (dénoté par N) de la table des faits est donné par l'équation suivante :

$$\mathbf{N} = \prod_{\mathbf{i=1}}^{\mathbf{g}} \mathbf{m_i} \tag{5}$$

où m_i représente le nombre de fragments de la table de dimensions D_i.

D'après l'équation 5, nous constatons que la fragmentation dérivée de la table des faits peut générer un très grand nombre de fragments et en conséquence beaucoup de sous-schémas en étoile.

Exemple 2 *Considérons le schéma en étoile de l'exemple 1 telles que les tables de dimensions soient partitionnées comme suit :*

- CLIENT *en 50 fragments en utilisant l'attribut "Etat"* [1],
- TEMPS *en 36 fragments en utilisant l'attribut "Mois", et*
- PRODUIT *en 80 fragments en utilisant l'attribut type de produit.*

[1] cas de 50 états aux U.S.A.

La table des faits est donc fragmentée en ***144000*** *(=* $\mathbf{5}0 \times 36 \times 80$*) fragments et le schéma en étoile en* ***144000*** *sous-schémas. Il devient difficile pour l'administrateur de l'entrepôt de gérer tous ces fragments.*

A travers l'exemple précédent, il apparaît indispensable de *réduire le nombre de fragments de la table des faits* pour une meilleure gestion. Il est donc nécessaire d'introduire des méthodes de sélection des tables de dimension à fragmenter pour :

- Éviter l'explosion du nombre de fragments de la table des faits.
- Garantir une meilleure performance d'exécution d'un ensemble de requêtes les plus fréquentes.

Pour atteindre le premier objectif, nous donnons à l'administrateur de l'entrepôt la possibilité de choisir le nombre maximum de fragments (W). Ce nombre est fixé en fonction du *coût de maintenance* de l'entrepôt fragmenté. Pour satisfaire le deuxième objectif, nous devons avoir la possibilité d'augmenter le nombre de fragments tant que la performance globale peut être améliorée. Le problème est donc de trouver un compromis entre le coût de maintenance et le coût d'exécution des requêtes, comme le montre la Figure 2.

Pour satisfaire ce compromis nous allons utiliser les algorithmes génétiques vu leurs capacité d'explorer un espace de solutions plus large [Bäck, 1995, Bennett *et al.*, 1991]. Notre problème ressemble au problème d'allocation de documents sur multiprocesseurs [Frieder et Siegelmann, 1997], et réplication des données [Loukopoulos et Ahmad, 2004], où les algorithmes génétiques ont donné des meilleurs résultats. Notons que ces algorithmes ont été également utilisés pour résoudre plusieurs problèmes d'optimisation des requêtes OLAP comme la sélection des vues matérialisées [Zhang et Yang, 1999].

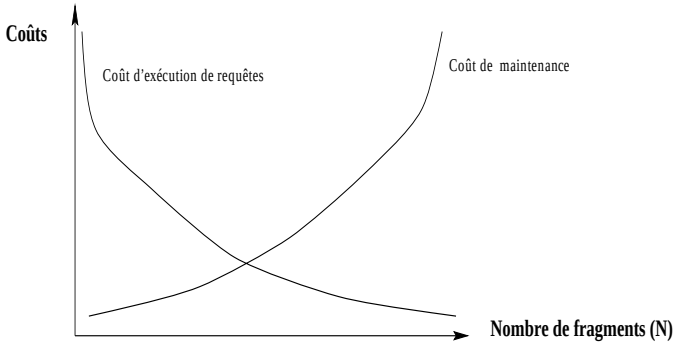

FIG. 2 – Evolution des coûts en fonction du nombre de fragments

Cet article est divisé en cinq sections. Après l'introduction du problème de la fragmentation dans le contexte des entrepôts de données modélisés par un schéma en étoile, nous présentons ses contraintes d'optimisation. La section 2 présente la complexité de la fragmentation horizontale dérivée de la table des faits. La section 3 présente l'algorithme génétique avec ses quatre étapes principales (sélection, codage, mutation et fonction sélective). Une étude expérimentale est réalisée en utilisant un benchmark

APB-1 release II (Section 4). La section 5 conclut le papier et préconise quelques voies de recherche.

3 Algorithme génétique

Les algorithmes génétiques (AG) sont des méthodes d'optimisation de fonctions [Bäck, 1995]. Ces algorithmes s'inspirent de l'évolution génétique des espèces, schématiquement, ils copient de façon extrêmement simplifiée certains comportements des populations naturelles. Ainsi, ces techniques reposent toutes sur l'évolution d'une population des solutions, qui, sous l'action de règles précises optimisent un comportement donné, exprimé sous une fonction, dite fonction sélection (fitness function) ou adaptation à l'environnement.

3.1 Concepts de base

Un AG est un algorithme itératif de recherche d'optimum, il manipule une population de taille constante. Cette dernière est formée de candidats appelés individus. La taille constante de la population entraîne un phénomène de compétition entre les individus. Chaque individu représente le codage d'une solution potentielle au problème à résoudre. Il est constitué d'un ensemble d'éléments appelés gènes, pouvant prendre plusieurs valeurs appartenant à un alphabet non forcément numérique [Bäck, 1995].

A chaque itération, appelée génération, est créée une nouvelle population avec le même nombre d'individus. Cette génération consiste en des individus mieux "adaptés" à leur environnement tel qu'il est représenté par la fonction sélective. Au fur et à mesure des générations, les individus vont tendre vers l'optimum de la fonction sélective. La création d'une nouvelle population, à partir de la précédente, se fait par application des opérateurs génétiques que sont : la sélection, le croisement et la mutation. Ces opérateurs sont stochastiques.

La sélection des meilleurs individus est la première opération dans un algorithme génétique. Au cours de cette opération, l'algorithme sélectionne les éléments pertinents qui optimisent le mieux la fonction. Le croisement permet de générer deux individus nouveaux "enfants" à partir de deux individus sélectionnés "parents" , tandis que la mutation réalise l'inversion d'un ou plusieurs gènes d'un individu [Bäck, 1995].

La Figure 3 illustre les différentes opérations qui interviennent dans un algorithme génétique : Les AGs fonctionnent sur une population d'individus N (la population représente tous les schémas de fragmentation possibles), que l'on croise entre eux pour trouver un individu "parfait" qui correspond au problème posé. Dans le cas de notre étude, le but est de trouver un schéma de fragmentation optimal de l'entrepôt de données.

Etant donné qu'un individu doit représenter une solution à part entière du problème posé, il devra donc représenter un schéma de fragmentation possible de l'entrepôt. Ce schéma de fragmentation doit être représenté par un individu unique qui représente un schéma de fragmentation possible d'un entrepôt de données.

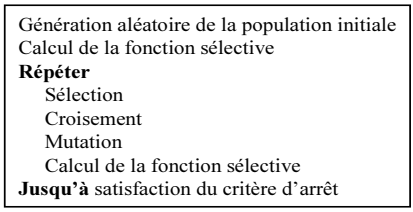

FIG. 3 – Algorithme génétique

3.2 Processus de création d'un individu

Tout algorithme de fragmentation horizontale nécessite un ensemble de requêtes les plus fréquentes. A partir de ces requêtes, nous extrairons deux types d'informations : *qualitative* et *quantitative*. Les informations qualitatives concernent les tables de dimension et sont représentées par les prédicats de sélection simples utilisés dans les requêtes. Les informations quantitatives concernent la sélectivité de ces prédicats et les fréquences d'accès des requêtes. Nous rappelons qu'un prédicat simple p est défini par

$$p : A_i\ \theta\ \text{Valeur} \quad ,$$

où A_i est un attribut d'une relation à fragmenter, $\theta \in \{=, <, \leq, >, \geq, \neq\}$, Valeur $\in Dom(A_i)$.

Avant de décrire le processus de codage des individus de la population, nous devons réaliser les tâches suivantes :

1. l'extraction de tous les prédicats de sélection simples utilisés par les n requêtes.
2. L'attribution à chaque table de dimension D_i $(1 \leq i \leq d)$ d'un *ensemble de prédicats simples* EPS^{D_i} qui lui est propre.
3. Toute table de dimension D_i ayant $EPS^{D_i} = \phi$, ne sera pas fragmentée. Soit $D_{candidat}$ l'ensemble de toutes les tables de dimension ayant leur ensemble de prédicats simples non vide. Soit g la cardinalité de l'ensemble $D_{candidat}$ $(g \leq d)$.
4. L'application de l'algorithme COM_MIN [Özsu et Valduriez, 1999] à chaque ensemble de prédicats simples d'une table de dimension D_i. Il fournit en sortie une forme complète et minimale de ces ensembles. La règle de complétude et de minimalité assure que si une table est fragmentée en au moins deux fragments, elle sera accédée différemment par au moins une application [Özsu et Valduriez, 1999].

3.2.1 Une représentation des fragments horizontaux

L'analyse des clauses représentant les fragments horizontaux permet d'effectuer une partition du domaine des attributs de fragmentation en sous-domaines appelés sous-domaines stables (SDS) [Simonet et Simonet, 1994]. Les éléments de cette partition sont déterminés par les prédicats de simples.

Exemple 3 *Pour montrer comment les prédicats de fragmentation définissent des partitions de chaque domaine des attributs de fragmentation (un attribut de*

fragmentation est un attribut qui participe dans le processus de partitionnement). Considérons trois attributs de fragmentation Age et Ville (de la table de dimension Client), Saison (de la table de dimension Temps). Supposons que les domaines des attributs de fragmentation sont :

- $Dom(Age) =]0, 120]$
- $Dom(Sexe) = \{`M', `F'\}$,
- $Dom(Ville) = \{$*"Grenoble"*, *"Paris"*$\}$.
- *Sur l'attribut Age, trois prédicats simples sont définis :* $p_1 : Age \leq 18$, $p_2 : Age \geq 60$, *et* $p_3 : 18 < Age < 60$. *Le domaine de l'attribut Age (*$]0, 120]$*) est donc partitionné en trois SDSs par les trois prédicats (*p_1, p_2, *et* p_3*) de la manière suivante (voir Figure 4) :*
  $Dom(Age) = d_{11} \cup d_{12} \cup d_{13}$, *avec* $d_{11} =]0, 18]$, $d_{12} =]18, 60[$, $d_{13} = [60, 120]$.
- *D'une manière similaire, le domaine de l'attribut Sexe est partitionné en deux SDSs :* $Dom(Sexe) = d_{21} \cup d_{22}$, *avec* $d_{21} = \{`M'\}$, $d_{22} = \{`F'\}$
- *Finalement, l'attribut Ville est partitionné en deux SDSs :* $Dom(Ville) = d_{31} \cup d_{32}$, *avec* $d_{31} = \{$*"Grenoble"*$\}$ *et* $d_{32} = \{$*"Paris"*$\}$.

Les différents SDSs de chaque attribut de fragmentation sont représentés par la Figure 4.

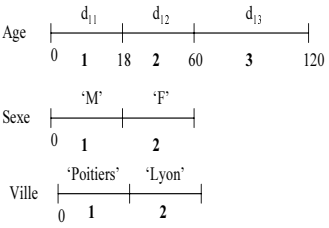

FIG. 4 – Les SDSs des attributs de fragmentation

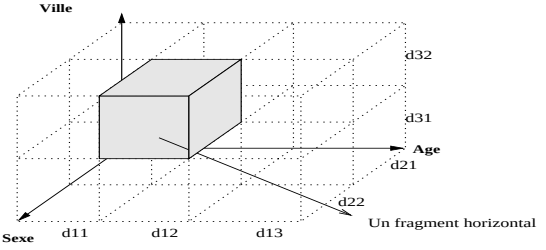

FIG. 5 – Un exemple de représentation de fragments dans le cas de trois attributs

Chaque attribut de fragmentation sera représenté par un tableau d'entier de n cellules, où n correspond au nombre de sous domaines, les valeurs dans les

cellules de ce tableau oscilleront entre 1 et n. Si deux cellules ont la même valeur, on regroupe les sous domaines de manière à n'en former qu'un.

Chaque individu (un schéma de fragmentation) est donc représenté par un tableau d'entier (voir Table 1) ou par un cube (voir Figure 5).

Sexe	1	1		
Saison	2	1	3	3
Age	2	1	2	

TAB. 1 – Un exemple d'un individu

A partir de cette table, on peut déduire que la fragmentation de l'entrepôt ne se fera pas sur l'attribut Sexe, car tous les sous domaine de l'attribut ont la même valeur. En conséquence, on fragmentera en utilisant l'attribut Saison et l'attribut Age. Pour l'attribut Saison, trois prédicats de fragmentation sont possibles, à savoir :

1. P_1 : Saison = "Printemps",
2. P_2 : Saison = "Eté",
3. P_3 : Saison = "Automne" $\vee$ Saison = "Hiver".

Pour l'attribut Age, deux prédicats sont possibles : $P_4 : Age \leq 18 \vee Age \geq 60$ et $P_5 : 18 < Age < 60$

A partir de ces prédicats, nous pouvons fragmenter le schéma de l'entrepôt en 6 fragments, où chaque fragment est défini par une clause de prédicats :

1. $P_1 \wedge P_4$;
2. $P_1 \wedge P_5$;
3. $P_2 \wedge P_4$;
4. $P_2 \wedge P_5$;
5. $P_3 \wedge P_4$;
6. $P_3 \wedge P_5$.

Notons que chaque fragment est représenté par une clause conjonctive de prédicats simples. Notons que tout algorithme de fragmentation doit satisfaire les règles de correction suivante [Özsu et Valduriez, 1999] :

- Complétude : elle assure que tous les n-uplets d'une relation sont associés à au moins un fragment.
- Disjonction : elle assure que les fragments d'une relation sont disjoints deux à deux
- Reconstruction : elle assure qu'une relation peut être reconstruite à partir de ses fragments.

Le codage de notre génome doit satisfaire les règles de correction. Pour cela, nous avons décidé de représenter un individu par un tableau d'entier pour chaque attribut, où une cellule correspondrait à un sous domaine de l'attribut.

3.2.2 Intérêt de ce codage

Notre codage présente plusieurs avantages :

1. les nouveaux individus issus du croisement sont compris dans la limite du domaine.
2. Il n'y a pas de chevauchement de fragments.
3. Tous les sous domaines sont représentés.
4. La mutation ne peut pas rendre un individu invalide

Cette représentation d'un individu permet de définir les fragments d'une table des faits ou bien d'une table dimension.

L'avantage d'une telle représentation est que, quelle que soit la façon dont est obtenue l'individu, elle est toujours valide dans les cas d'une génération aléatoire ou un croisement. Eviter de vérifier à chaque création d'un individu s'il est valide ou non représente un gain de temps en calcul.

3.3 Sélection des individus

Nous utilisons la méthode de la roulette pour sélectionner nos individus parents. Dans cette méthode chaque individu est accompagné de son évaluation. Plus cette évaluation est élevée, plus l'individu a de chances d'être sélectionné. Les individus choisis sont appelés le "père" et la "mère".

- Une fois les individus choisis, on détermine si on va les croiser ou non. On détermine si on croise ou pas en tirant un chiffre aléatoire ;
- Si ce chiffre est supérieur à un certain taux de croisement (en général fixé entre 70 et 90 %), on ne croise pas et les individus sont réinjectés directement dans la population de génération suivante.
- Si le père et la mère sont trop semblables, un seul sera réinjecté.

Dans l'autre cas, on crée deux enfants à partir des deux parents. On ajoute alors ces enfants à la population.

3.4 Types de croisements

Plusieurs types de croisement peuvent être envisagés : le croisement mono-point (1 seul point de croisement par individu), le croisement multipoints (2 ou plusieurs points de croisement par individu), le croisement uniforme (opération bit à bit entre les individus).

Nous avons choisi d'utiliser un croisement multipoints pour la raison suivante : nous savons depuis la création d'un individu que les prédicats sont placés les uns à la suite des autres dans le vecteur d'entiers. Les individus sont croisés une fois sur chaque prédicat, pour ne pas désavantager le croisement d'un prédicat par rapport à un autre. On sait en effet que chaque prédicat dispose d'un nombre d'entiers différents. Si on n'effectue qu'un croisement par individu le prédicat qui a un grand nombre d'entiers (ex : ville : autant d'entiers que de villes) aura une probabilité de croisement supérieure à un prédicat qui n'a que quelque entiers (sexe : masculin-féminin donne 2 entiers). Cette opération de sélection est effectuée tant que la population n'a pas atteint le nombre convenable d'individus.

3.5 Evaluation de l'individu (la fonction sélective)

L'évaluation d'un individu permet de définir quel individu est meilleur par rapport à un autre. Cette évaluation donne un pourcentage qui est la somme de deux paramètres : le respect du seuil représentant le nombre de fragments et la rapidité des requêtes. Un nombre de point modifiable est alloué à ces deux paramètres.

- *respect du seuil* : par défaut 55 points sur cent sont alloués. Si le nombre de fragments obtenus est égale à plus ou moins cinq pour cent le nombre de fragments préconisé par l'administrateur (ou seuil), tous les points sont alloués. Par la suite plus on s'éloigne du seuil, moins de points sont ajoutés au résultat.
- *rapidité des requêtes* : un nombre de points uniforme est alloué à chaque requête. Par défaut, comme il reste 45 points (100 - 55 du seuil) et que nous avons 15 requêtes, 3 points sont alloués à chaque requête. La rapidité d'une requête dépend d'un calcul selon un modèle de coût qui nous donne le nombre d'entrées sorties qu'effectue la requête. Ce modèle est expliqué ci-dessous. Comme précédemment pour le seuil, on donne tous les points (ici 3) si le nombre d'entrées-sorties est inférieur à un certain chiffre. Puis, plus le nombre d'entrées-sorties augmente, moins on donne de points, selon une fonction linéairement décroissante. Quand le nombre d'entrées-sorties pour une requête est trop élevé, aucun point n'est attribué.

Les performances des requêtes dépendent du transfert des données de la mémoire secondaire (disque dur) vers la mémoire vive. Nous supposons que les tables de dimension résident en mémoire.

Notons que chaque prédicat de fragmentation p_i d'une table de dimension D_j est associé à un facteur de sélectivité $Sel_{D_j}^{p_i}$ (compris entre 0 et 1). Ce prédicat p_i a également un facteur de sélectivité sur la table des faits noté par $Sel_F^{p_i}$.

Pour exécuter une requête Q sur un schéma d'un entrepôt de données fragmenté en N sous-schémas de l'entrepôt $\{SED_1, SED_2, ..., SED_N\}$, nous devons identifier le(s) sous-schéma(s) valide(s). Pour réaliser cette identification, une variable booléenne dénotée par $valide(Q, SED_i)$ est définie :

$$valide(Q, SED_i) = \begin{cases} 1 & \text{si le sous-schéma } SED_i \text{ est valide pour la requête } Q \\ 0 & \text{sinon} \end{cases}$$

Pour exécuter une requête Q sur un schéma d'un entrepôt de données fragmenté en N, nous proposons un modèle de coût donné par l'équation suivante :

$$Nombre\_E/S = \sum_{j=1}^{N} valide(Q, SDE_j) \prod_{i=1}^{M_j} \frac{(Sel_F^{p_i} \times ||F|| \times L)}{PS} \quad (6)$$

avec, $||F||$ = le nombre d'enregistrements de la table des faits, L la taille d'un enregistrement, M_j représente le nombre de prédicats de sélection définissant la clause d'un fragment de la table des faits du schéma SDE_j et PS la taille d'une page disque.

Dans cette étude, nous avons utilisé la sélectivité des prédicats dans la table des faits et non dans les tables dimensions. La sélectivité peut se faire de manière uniforme, on parle alors de répartition uniforme (RU) et de manière non uniforme (RNU) par une initialisation des facteurs de sélectivité.

3.6 La mutation

L'analogie avec le monde vivant nous amène à nous poser la question de la mutation. Dans la nature, les mutations sont fréquentes et entraînent des variations plus ou moins marquées de l'individu. Dans notre modèle nous avons choisi un taux de mutation qui se situe entre 30% et 6%, c'est le taux habituellement utilisé. Il faut choisir un juste milieu entre une trop faible et une trop forte mutation. Les mutations s'effectuent aux niveau des attributs de notre individu. Suivant un nombre aléatoire, nous choisissons de muter un ou plusieurs attributs de l'individu. On peut voir sur le schéma suivant la mutation d'un attribut.

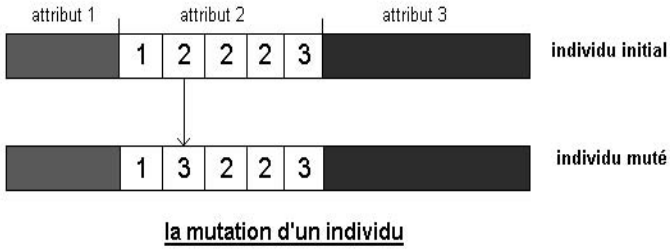

FIG. 6 – Un exemple de mutation

Sur l'exemple ci-dessus figure la mutation d'un individu sur l'un de ses attributs. On remarque qu'il y avait trois fragments dans l'individu initial en ce qui concerne l'attribut 2. Sur l'individu muté, on remarque qu'il y a toujours trois fragments mais les intervalles pris en compte ne sont plus les mêmes. Dans la pratique, il pourrait y avoir plus d'intervalles distincts ou un regroupement d'intervalles. De même, les mutations pourraient se produire sur plusieurs attributs de l'individu.

4 Evaluation de notre approche

Le moteur génétique a été développé en Visual C sur une machine Pentium 4 fréquence 2.80 Ghz et une Ram de 256 Mo. L'architecture de notre application est décrite par la Figure 7. Afin de valider notre étude, nous avons utilisé le benchmark APB-1 release II [Council, 1998]. Ce benchmark utilise un schéma en étoile décrit dans la Figure 8. Il comporte quatre tables de dimensions (Prodlevel, Custlevel, Timelevel et Chanlevel) et une table des faits (Actvars). Les caractéristiques de chaque table sont représentées dans le table 2.

On a utilisé 15 requêtes décisionnelles. Chaque requête comporte des prédicats de sélections (chacun ayant un facteur de sélectivité). Chaque requête a une fréquence d'accès. Nous avons utilisé un taux de croisement de 70% et un taux de mutation : variable de 30% au début jusqu'à 6% à la fin, décrémenté chaque 50 générations. Le but visé est d'être permissif au début (taux de mutation élevé) pour pouvoir explorer

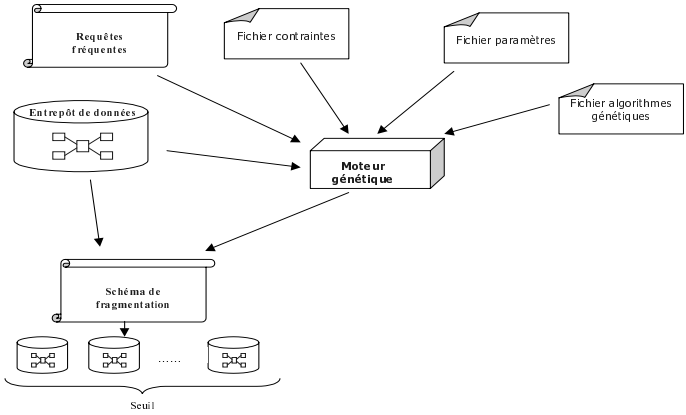

FIG. 7 – Architecture de moteur génétique

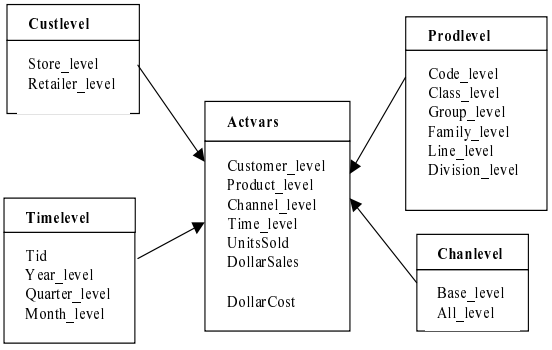

FIG. 8 – Le schéma du banc d'essai (APB-1)

plus de points dans l'espace de recherche. Après plusieurs générations, un petit taux de mutation (6%) est exigé pour éviter de rendre la recherche très aléatoire. Nous avons utilisé 1 500 générations, dont 40 individus par génération. 9 attributs de fragmentation sont considérés.

Si l'administrateur décide de fixer un seuil très grand Seuil = 2000, les tables de dimension sont fragmentées de la manière suivante comme le montre la Figure 10.

La meilleure façon de fragmenter l'entrepôt est la suivante :

1. la table Prodlevel en 48 fragments,
2. la table Timelevel en 7 fragments,
3. la table Custlevel en 2 fragments,

Table	Nombre d'enregistrements	Taille d'un enregistrement
Actvars	24786000	74
Chanlevel	9	24
Custlevel	900	24
Prodlevel	9000	72
Timelevel	24786000	36

TAB. 2 – Tailles des tables

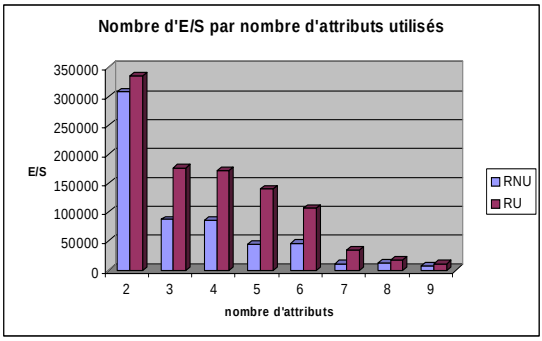

FIG. 9 – Nombre d'E/S par rapport au nombre d'attributs utilisés

Code_level	0	1	2	0									**3 Fragments Prodlevel**
Group_level	0	1											**2 Fragments Prodlevel**
Family_level	1	2	0	2	3								**4 Fragments Prodlevel**
Line_level	0	0											**1 Fragment Prodlevel**
Division_level	1	1	0	0									**2 Fragments Prodlevel**
Year_level	0	0											**1 Fragment Timelevel**
Month_level	5	10	7	4	10	2	4	0	3	7	4	2	**7 Fragments Timelevel**
Retailer_level	1	1	1	2									**2 Fragments Custlevel**
All_level	1	0	2	0	2								**3 Fragments Chanlevel**

FIG. 10 – La Structure des fragments finaux

4. la table Chanlevel en 3 fragments,
5. et table des faits en 2016 fragments

La Figure 9 montre l'évolution du nombre d'E/S par rapport au nombre d'attributs

utilisés dans la fragmentation. Les résultats montrent que le nombre d'attributs utilisés dans le processus de fragmentation a un impact très important dans la réduction du nombre d'E/S total. On remarque aussi que la répartition non uniforme donne de meilleurs résultats par rapport à la répartition uniforme.

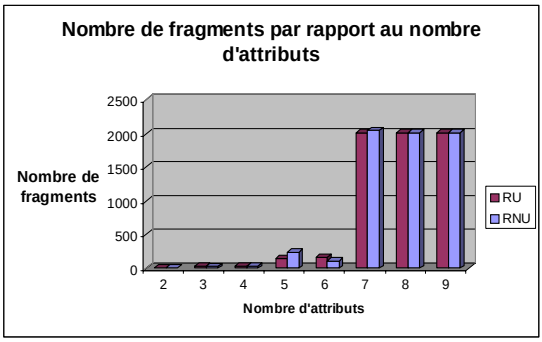

FIG. 11 – Nombre de fragments par rapport au nombre d'attributs utilisés

La Figure 11 montre l'évolution du nombre de fragments de la table des faits avec l'augmentation des attributs utilisés dans la fragmentation. On constate que le type de répartition n'a pas d'effet considérable sur le nombre total de fragments.

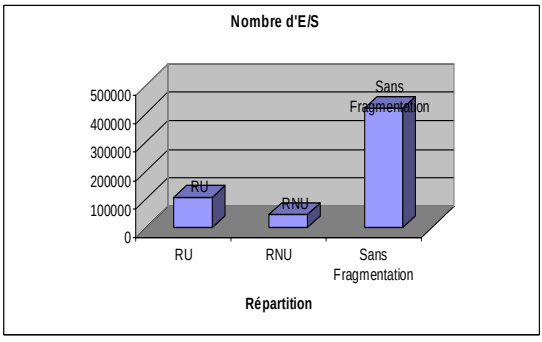

FIG. 12 – Nombre d'E/S total par rapport à la répartition

La Figure 12 montre clairement que la fragmentation diminue largement le nombre d'E/S total d'exécution des requêtes ce qui confirme la théorie de la fragmentation. On constate aussi que la répartition non uniforme est plus bénéfique que la répartition uniforme.

La Figure 13 montre que le nombre de tables de dimensions fragmentées joue un rôle très important dans la réduction du nombre d'E/S. Tant que le nombre de tables

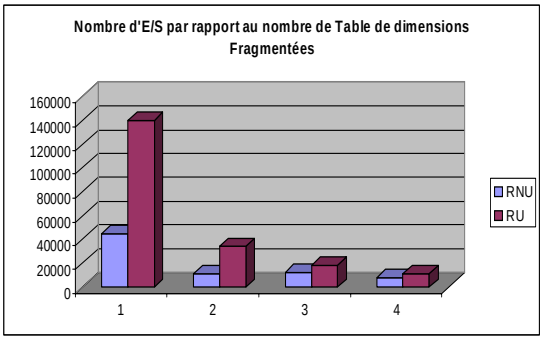

FIG. 13 – Nombre d'E/S par rapport au nombre de tables de dimensions fragmentées

de dimension augmente, la performance de requêtes également. La Figure 14 montre

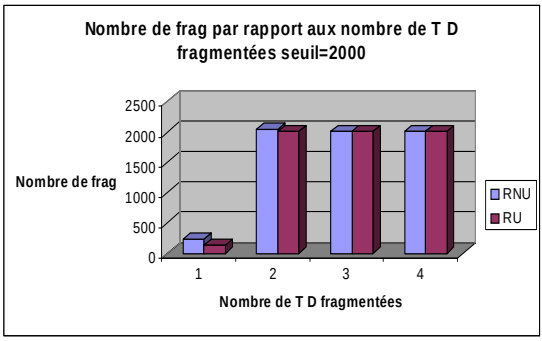

FIG. 14 – Nombre de fragments vs. nombre de tables de dimensions fragmentées

que le nombre de fragments augmente avec l'augmentation du nombre de tables de dimensions à fragmenter mais notre algorithme permet de contrôler cette augmentation pour donner un nombre de fragments qui ne viole pas la contrainte du seuil.

Dans la Figure 15 on constate que le rapport entre le nombre de fragments retourné par l'algorithme et le nombre de fragments possibles varie selon le nombre de fragments utilisés. Lorsqu'on utilise moins de six attributs, ce rapport est grand (plus de 35%) car le nombre de fragments possible est petit. A partir de six attributs utilisés, l'algorithme retourne un rapport minime jusqu'à arriver à un rapport très petit (moins de 2%) lorsqu'on utilise neuf attributs. Cela est dû d'un côté à l'augmentation du nombre de fragments possibles (par l'augmentation du nombre d'attributs) et de l'autre coté, le nombre de fragments retournés se stabilise au voisinage du seuil. La Figure 16 représente l'évolution du nombre d'E/S par rapport au seuil. On constate

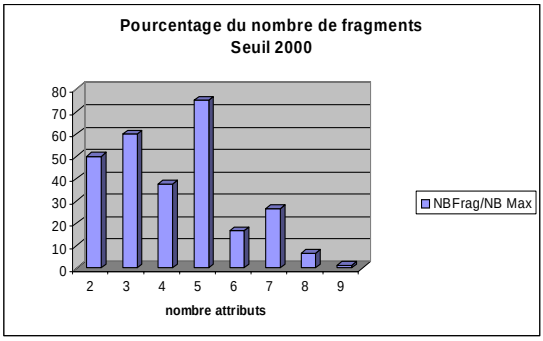

FIG. 15 – Nombre de fragments vs. nombre de fragments possibles

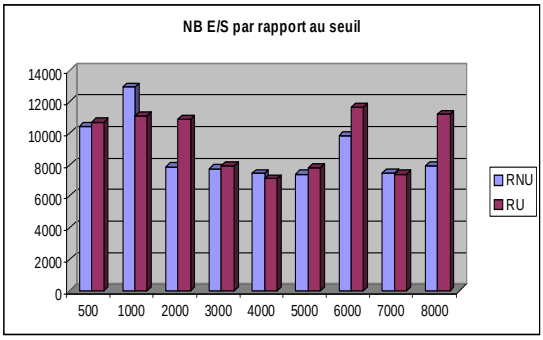

FIG. 16 – Evolution du nombre d'E/S par rapport au seuil

qu'un seuil de 4000 est le plus adapté. Il représente 2,6% du nombre total de fragments.

La Figure 17 montre que le nombre de fragments retournés par l'algorithme augmente toujours avec l'augmentation du seuil mais reste toujours au voisinage de ce dernier.

Dans les figures qui suivent 18, 19 et 20 on a changé la sélectivité des prédicats pour voir l'effet de la sélectivité sur le nombre total d'E/S et le nombre de fragments.

On constate que lorsqu'on augmente la sélectivité des prédicats, le nombre total d'E/S augmente. Cela est dû au fait qu'une forte sélectivité implique un nombre d'enregistrements satisfaisant le prédicat plus grand. Ce qui conduit à un nombre d'E/S plus important. On constate aussi que la sélectivité n'a pas d'effet important sur le nombre de fragments total.

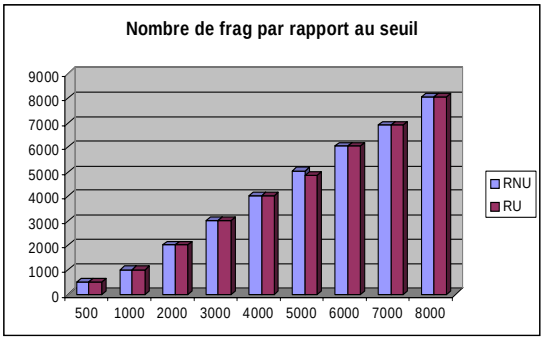

FIG. 17 – Evolution du nombre de fragments par rapport au seuil

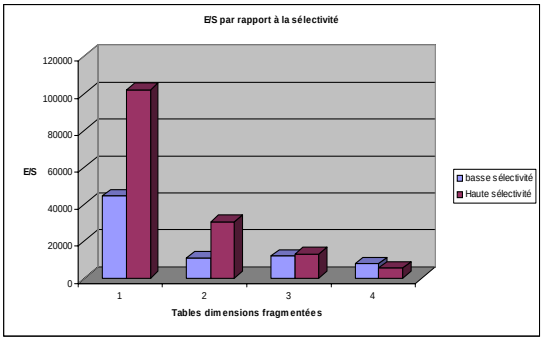

FIG. 18 – Nombre d'E/S par rapport à la sélectivité et le nombre de tables fragmentées

5 Conclusion

Dans ce papier nous avons formalisé le problème de la fragmentation horizontale dans les entrepôts de données modélisés par des schémas en étoile. Nous avons d'abord développé une méthodologie de fragmentation pour la table des faits, en adaptant les techniques utilisées dans les bases de données relationnelles. Par la suite, nous avons présenté les problèmes liés à la fragmentation de la table des faits en fonction des tables de dimensions et nous avons suggéré l'utilisation des algorithmes génétiques. Un processus de codage des schémas de fragmentation possibles est décrit. Un modèle de coût qui représente la fonction sélective a été développé afin de mesurer la qualité de la solution choisie. Finalement des expériences ont été conduites et ont montré qu'avec une fragmentation adéquate des tables de dimensions, il est possible d'améliorer les performances d'exécution des requêtes décisionnelles en respectant le seuil des fragments que l'administrateur doit choisir.

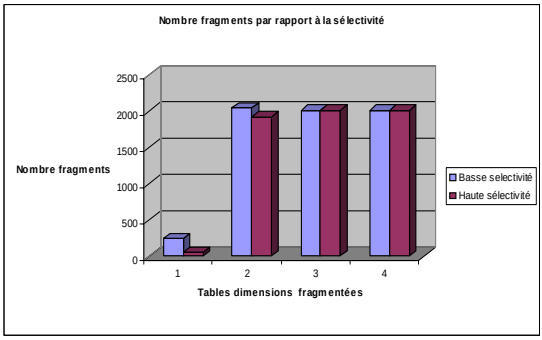

FIG. 19 – Nombre de fragments par rapport à la sélectivité et le nombre de tables fragmentées

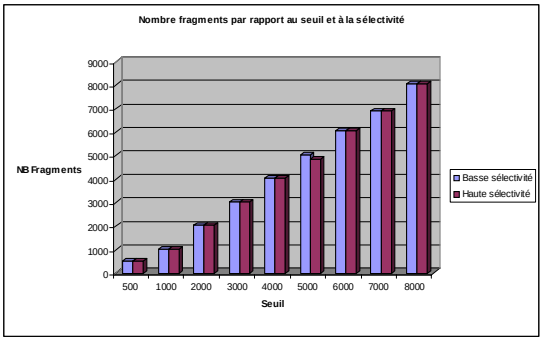

FIG. 20 – Evolution du nombre de fragments par rapport à la sélectivité et au seuil

Comme perspective il serait donc intéressant de développer ou d'aménager notre algorithme génétique pour tenir compte de l'évolution des requêtes tant dans leurs structures que dans leurs fréquences.

Références

[Bäck, 1995] T. Bäck. *Evolutionnary algorithms in theory and practice.* Oxford University Press, New York, 1995.

[Bellatreche *et al.*, 2004] L. Bellatreche, M. Schneider, H. Lorinquer, et M. Mohania. Bringing together partitioning, materialized views and indexes to optimize performance of relational data warehouses. *Proceeding of the International Conference on*

Data Warehousing and Knowledge Discovery (DAWAK'2004), pages 15–25, September 2004.

[Bennett *et al.*, 1991] K. P. Bennett, M. C. Ferris, et Y. E. Ioannidis. A genetic algorithm for database query optimization. *in Proceedings of the 4th International Conference on Genetic Algorithms*, pages 400–407, July 1991.

[Council, 1998] OLAP Council. Apb-1 olap benchmark, release ii. *http ://www.olapcouncil.org/research/bmarkly.htm*, 1998.

[Frieder et Siegelmann, 1997] O. Frieder et H.T. Siegelmann. Multiprocessor document allocation : A genetic algorithm approach. *IEEE Transactions on Knowledge and Data Engineering*, 9(4) :640–642, July 1997.

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

[Kotidis et Roussopoulos, 1999] Y. Kotidis et N. Roussopoulos. Dynamat : A dynamic view management system for data warehouses. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 371–382, June 1999.

[Loukopoulos et Ahmad, 2004] T. Loukopoulos et I. Ahmad. Static and adaptive distributed data replication using genetic algorithms. *in Journal of Parallel and Distributed Computing*, 64(11) :1270–1285, November 2004.

[Noaman et Barker, 1999] A. Y. Noaman et K. Barker. A horizontal fragmentation algorithm for the fact relation in a distributed data warehouse. *in the 8th International Conference on Information and Knowledge Management (CIKM'99)*, pages 154–161, November 1999.

[Özsu et Valduriez, 1999] M. T. Özsu et P. Valduriez. *Principles of Distributed Database Systems : Second Edition*. Prentice Hall, 1999.

[Sanjay *et al.*, 2004] A. Sanjay, V. R. Narasayya, et B. Yang. Integrating vertical and horizontal partitioning into automated physical database design. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 359–370, June 2004.

[Simonet et Simonet, 1994] A. Simonet et M. Simonet. Objects with views and constraints : From databases to knowledge bases. *in the Proceeding of the International Conference on Object Oriented Information Systems, OOIS'94*, pages 182–195, December 1994.

[Zhang et Yang, 1999] C. Zhang et J. Yang. Genetic algorithm for materialized view selection in data warehouse environments. *Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'99)*, pages 116–125, September 1999.

[Zilio *et al.*, 2004] D. C. Zilio, J. Rao, S. Lightstone, G. M Lohman, A. Storm, C. Garcia-Arellano, et S. Fadden. Db2 design advisor : Integrated automatic physical database design. *Proceedings of the International Conference on Very Large Databases*, pages 1087–1097, August 2004.

Summary

The problem of selecting an optimal fragmentation schema of a data warehouse for query optimization is more challenging compared to that in relational and object databases. This challenge is due to the several choices of partitioning star or snowflake schemas. Data partitioning is beneficial if and only if the fact table is fragmented based on the partitioning schemas of dimension tables. This may increase the number of fragments of the fact tables dramatically and makes their maintenance very costly. Therefore, the right selection of fragmenting schemas is important for better performance of OLAP queries. In this paper, we present a genetic algorithm for schema partitioning selection problem. The proposed algorithm gives better solutions since the search space is constrained by the schema partitioning. We conduct several experimental studies using the APB-1 release II benchmark for validating the proposed algorithm.