Combinaison des algorithmes génétique et de recuit simulé pour la conception physique des entrepôts de données

Kamel Boukhlafa* — **Ladjel Bellatreche ****

** Université de Laghouat, Algérie*
k.boukhalfa@mail.lagh-univ.dz
***LISI/ENSMA - 86960 Futuroscope Cedex France*
bellatreche@ensma.fr

RÉSUMÉ. *La fragmentation de données est une des techniques utilisée dans la conception physique des entrepôts de données. Elle permet d'accélérer l'exécution des requêtes OLAP et de faciliter la gestion des données de l'entrepôt. Pour fragmenter un entrepôt de données relationnel, la meilleure manière consiste d'abord à décomposer les tables de dimension ensuite à utiliser leurs schémas de fragmentation pour partitionner la table de faits. L'espace de recherche pour sélectionner le schéma de fragmentation optimal peut être très important. Dans ce papier, nous formalisons d'abord le problème de sélection d'un schéma de fragmentation pour un entrepôt de données relationnel comme un problème d'optimisation avec une contrainte de maintenance. Nous proposons ensuite une méthode hybride combinant un algorithme génétique et un algorithme de recuit simulé pour résoudre ce problème. Enfin, des expérimentations basées sur le benchmark APB-1 release II sont présentées afin de valider nos algorithmes.*

ABSTRACT. *Data partitioning is one of the physical data warehouse design techniques that accelerates OLAP queries and facilitates the warehouse manageability. To partition a relational warehouse, the best way consists in fragmenting dimension tables then using their fragmentation schemas to partition the fact table. The search space for selecting an optimal fragmentation schema in the relational data warehouse context may be exponentially large. In this paper, the horizontal fragmentation selection problem is formalised as an optimisation problem with a maintenance constraint. We propose, a hybrid method combining a genetic algorithm and a simulated annealing algorithm. Our experimental studies are based on APB-1 release II benchmark in order to validate our proposed algorithms.*

MOTS-CLÉS : *Algorithme génétique, Recuit simulé, Entrepôts de données, Fragmentation*

KEYWORDS: *Genetic algorithms, Simulated annealing, Data warehouse, Data partitioning*

1. Introduction

Un entrepôt de données stocke de grosses quantités de données consolidées et historisées. Il est particulièrement conçu pour répondre aux requêtes décisionnelles complexes. Un entrepôt de données relationnel est habituellement modélisé par un schéma en étoile (ou un schéma en flocon de neige) qui est caractérisé par une table des faits de taille importante et un certain nombre de tables de dimension plus petites. Sur ce schéma, des requêtes en étoile sont exécutées. La principale caractéristique de ces requêtes est qu'elles imposent des restrictions sur les valeurs des tuples des tables de dimension utilisées pour sélectionner des faits spécifiques ; ces faits seront groupés et agrégés selon les demandes des utilisateurs. Le goulot d'étranglement principal en évaluant de telles requêtes est de joindre la tables des faits avec les tables de dimension (Stöhr et al., 00). Plusieurs techniques ont été proposées pour optimiser ces requêtes ; nous pouvons ainsi citer : les vues matérialisées (Yang *et al.,* 1997), les index avancés (Chaudhuri *et al.,* 1998), la fragmentation horizontale (Sanjay *et al.,* 2004), et le traitement parallèle (Stöhr *et al.,* 2000).

La fragmentation horizontale constitue un aspect important dans la conception physique des bases de données (Sanjay *et al.,* 2004), (Papadomanolakis *et al.,* 2004). Elle a un impact significatif sur l'exécution des requêtes OLAP et la gestion de l'entrepôt de données. Dans le contexte des entrepôts de données relationnels, elle permet de partitionner les tables, les index et les vues matérialisées en plusieurs ensembles de lignes disjoints, physiquement stockés et séparément consultés (Sanjay et al., 04). Contrairement aux vues matérialisées et aux index, la fragmentation ne duplique pas les données, en conséquence elle réduit le coût de mises à jour (Papadomanolakis *et al.,* 2004). Elle peut également être combinée avec d'autres structures d'optimisation comme les index, les vues matérialisées (Bellatreche et al., 04), et le traitement parallèle (Stöhr *et al.,* 2000), (Rao *et al.,* 2002). Plusieurs travaux et systèmes commerciaux ont montré son utilité et son impact dans l'optimisation des requêtes OLAP (Sanjay *et al.,* 2004), (Kalnis *et al.,* 2001), (Stöhr *et al.,* 2000). Mais peu de travaux ont formalisé le problème de sélection d'un schéma de fragmentation horizontale pour accélérer l'exécution d'un ensemble de requêtes, et proposé des algorithmes de sélection. Dans (Bellatreche *et al.,* 2005), nous avons proposé une technique pour fragmenter la table des faits en fonction des schémas de fragmentation des tables de dimension. Concrètement, elle consiste d'abord à partitionner certaines/toutes les tables de dimension en utilisant leurs prédicats de sélection simples, ensuite à fragmenter la table de faits en utilisant les schémas de fragmentation des tables de dimension. Cette procédure de fragmentation prend en compte la caractéristique des requêtes en étoile entre la table des faits et les tables de dimension.

Pour illustrer cette procédure de fragmentation, nous considérons un schéma en étoile avec trois tables de dimension : *Client*, *Temps* et *Produit* et une table de fait *Ventes*. Supposons que la table de dimension *Client* est fragmentée en deux fragments *Client1* et *Client2* définis par les clauses suivantes : $Client_1 = \sigma_{Sexe='M'}(Client)$ et $Client_2 = \sigma_{Sexe='F'}(Client)$. Ce type de fragmentation est supporté par les SGBDs commerciaux. Cette fragmentation est matérialisée par la clause SQL suivante :

CREATE TABLE Client (NC number(4), Cnom varchar2(14), Sexe varchar2(1)) PARTITION BY LIST (Sexe) (PARTITION Client1 values ('M'), PARTITION Client2 values ('F')) ;

La table de faits Ventes peut être fragmentée en deux fragments Ventes1 et Ventes2 définis par : $Ventes_1 = Ventes \ltimes Client_1$ et $Ventes_2 = Ventes \ltimes Client_2$, où $\ltimes$ représente l'opération de semi-jointure.

Ce processus de fragmentation génère deux sous schémas en étoile S1 et S2 tels que : S1 : (Ventes1, Client1, produit, temps) (activités de ventes réalisées par les clients masculins seulement) et S2 : (Ventes2, Client2, produit, temps) (activités de ventes réalisées par les clientes seulement).

Le nombre de fragments horizontaux de la table de faits (noté N) généré par cette procédure de fragmentation est donné par : $\mathbf{N} = \prod_{\mathbf{i=1}}^{\mathbf{g}} \mathbf{m_i}$, où m_i et g représentent respectivement le nombre de fragments de la table de dimension D_i, et le nombre de tables de dimension participant dans le processus de fragmentation. Ce nombre peut être très grand (Bellatreche *et al.*, 2005). En conséquence, au lieu de gérer un seul schéma en étoile, l'administrateur de l'entrepôt de données (DWA) doit gérer N sous schémas en étoile. Il lui sera très difficile de maintenir tous ces sous schémas.

La fragmentation peut améliorer l'exécution de la jointure entre la table des faits et les tables de dimension. Si par exemple, l'administrateur veut exécuter une requête OLAP avec des opérations d'agrégation ne concernant que les clientes, la jointure entre table des faits Ventes et la table de dimension Client se traduit par une jointure entre Vente2 et Client2. Parallèlement, la fragmentation pourrait *être concurrente dans l'utilisation aux index de jointure binaires* (bit map join indexes). L'index de jointure est construit en traduisant les restrictions à la valeur de la colonne de la table de dimension. Nous pouvons construire un index de jointure binaire afin d'optimiser la requête précédente :

CREATE BITMAP INDEX IJB
ON *Client(Sexe)*
FROM Client, Ventes
WHERE Client.NC = Ventes.NC ;

La fragmentation horizontale dans les entrepôts de données relationnels est plus difficile que celle dans les bases de données relationnelles et objets. Pour fragmenter une table de faits, le DWA doit exécuter les étapes suivantes : (1) sélectionner la (les) table(s) de dimension participant dans le processus de fragmentation de la table de faits, (2) décomposer ces tables de dimension, et (3) utiliser leurs schémas pour fragmenter la table de faits. Pour répondre au problème de la fragmentation horizontale dans le contexte des entrepôts de données, nous proposons une approche qui combine un algorithme génétique et un recuit simulé (Bäck, 1995), (Bennett *et al.*, 1991).

Dans (Bellatreche *et al.*, 2005), nous avons proposé un algorithme génétique pour résoudre ce problème, mais sans tenir compte de la contrainte de maintenance dans le calcul de la fonction d'évaluation (fitness function).

Ce papier est divisé en six sections : la section 2 formalise le problème de sélection d'un schéma de fragmentation dans les entrepôts de données modélisés par un schéma en étoile comme un problème d'optimisation. La section 3 décrit notre algorithme génétique utilisé pour résoudre notre problème de fragmentation en décrivant en détail la fonction d'évaluation et la pénalité. La section 4 présente l'algorithme du recuit simulé. La section 5 montre les résultats expérimentaux obtenus en utilisant le benchmark APB-1 release II. La section 6 conclut le papier en récapitulant les résultats principaux et en suggérant des travaux futurs.

2. Problème de sélection d'un schéma de fragmentation horizontale

Le problème de fragmentation horizontale peut être formalisé de la façon suivante : étant donné (1) un ensemble de tables de dimension $D = \{D_1, D_2..., D_d\}$ et une table de fait F, (2) un ensemble de requêtes OLAP fréquentes $Q = \{Q_1, Q_2..., Q_m\}$, où chaque requête $Q_i (1 \leq i \leq m)$ possède une fréquence d'accès, et (3) un seuil (W fixé par le DWA) représentant le nombre maximum de fragments qu'il peut maintenir. Notons que la satisfaction de cette contrainte évite une explosion du nombre de fragments de la table de faits.

Le problème de fragmentation horizontale consiste à (1) déterminer un ensemble de tables de dimension à fragmenter, et (2) utiliser leurs schémas de fragmentation pour fragmenter la table de faits F en un ensemble de N fragments horizontaux (appelés les fragments de faits) tels que le coût total d'exécution des requêtes sur le schéma en étoile fragmenté soit réduit, et la contrainte de maintenance soit satisfaite ($N \leq W$).

Notons que le nombre de schémas de fragmentation possibles d'un entrepôt de données peut être très grand. Notre approche utilise d'abord un algorithme génétique (AG) et ensuite un algorithme de recuit simulé (ARS). Les AGs ont été largement utilisés pour la conception physique des bases de données. On peut citer, le problème d'optimisation des requêtes de jointure (Ioannidis et al., 90), le problème de sélection des vues matérialisées (Yu *et al.,* 2004) et l'automatisation de la conception physique des bases de données parallèles (Rao *et al.,* 2002). Puisque les AG ont contribué à l'optimisation de l'opération de jointure, et en raison de la similitude entre le problème de sélection des index et le problème de fragmentation horizontale dans le contexte d'entrepôt de données (Bellatreche *et al.,* 2004) nous les avons utilisés pour notre problème.

Les AGs (Holland, 1975) sont des méthodes de recherche basées sur le concept évolutionnaire de la mutation naturelle et la survie des individus les plus convenables. Ils appliquent trois opérations génétiques de recherche, à savoir, la sélection, le croisement, et la mutation, pour transformer une population initiale de chromosomes, dans l'objectif d'améliorer leur qualité. Le concept fondamental lié à la structure des AGs est la notion du chromosome, qui est une représentation codée d'une solution possible. Avant que le processus de recherche commence, un ensemble de

chromosomes est initialisé pour former la première génération. Les trois opérations de recherche génétiques sont appliquées à plusieurs reprises, afin d'obtenir une population avec de meilleures caractéristiques. La forme générale d'un AG est la suivante :
Génération de la population initiale
Sélection
Tant que condition d'arrêt non atteinte
    Croisement
    Mutation
    Sélection
Fin Tant que

D'autre part, le recuit simulé évite le phénomène de la convergence prématurée inhérent aux AGs. Le recuit simulé (Kirkpatrick *et al.,* 1983) est une technique aléatoire pour trouver une solution proche de l'optimale des problèmes d'optimisation combinatoire. Il commence par une solution candidate aléatoire, ensuite il essaye à plusieurs reprises de trouver une solution meilleure en se déplaçant vers un voisin avec une fitness plus élevée, jusqu'à l'obtention de la meilleure solution (ayant la meilleure fitness). Pour éviter de se bloquer sur un optimum local, le RS, permet, occasionnellement, des déplacements vers des solutions avec une fitness inférieure en utilisant un paramètre température pour contrôler l'acceptation des déplacements avec une certaine probabilité dépendant d'un certain nombre de paramètres. Ces deux algorithmes seront décrits en détails dans les sections suivantes.

3. L'algorithme génétique

La partie la plus difficile dans l'application des AGs est la représentation des solutions qui représentent des schémas de fragmentation. Traditionnellement, les AGs utilisent la représentation binaire des chromosomes. Dans notre étude, nous proposons une autre représentation qui nécessite quelques explications.

3.1. *Mécanisme de codage*

Notons que tout fragment horizontal est défini par une clause de conjonction de prédicats définis sur les attributs de fragmentation (Bellatreche *et al.,* 2005). Avant de présenter notre mécanisme de codage des fragments des tables de dimension, nous devons identifier quelles tables de dimension participant dans la fragmentation de la table des faits. Pour se faire, nous procédons de la façon suivante : (1) extraire tous les attributs simples employés par les m requêtes, (2) assigner à chaque table de dimension $D_i (1 \leq i \leq d)$ son ensemble de prédicats simples, noté par SPD_i, (3) les tables de dimension ayant un SSPD vide ne sont pas prises en considération dans le processus de fragmentation.

Notons que chaque attribut de fragmentation possède un domaine de valeurs. Les clauses des fragments horizontaux définissent le découpage du domaine de chaque attribut en plusieurs sous domaines. Considérons deux attributs de fragmentation Age et Sexe de la table de dimension Client et un attribut Saison de la table de dimension Temps. Les domaines de ces attributs sont définis comme suit : $Dom(Age) =]0, 120]$, $Dom(Sexe) = \{`M', `F'\}$, et
$Dom(Saison) = \{$"Et'', "$Printemps''$, "$Automne''$, "$Hiver''\}$. Supposons que, sur l'attribut Age, trois prédicats simples sont définis comme suit : p_1 : (Age ≤ 18), p_2 : ($Age \geq 60$), et p_3 : ($18 < Age < 60$). Le domaine de cet attribut est subdivisé en trois sous domaines : 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 décomposé en deux sous domaines, Dom(Sexe) = $d_{21} \cup d_{22}$, avec $d_{21} = \{`M'\}$, $d_{22} = \{`F'\}$. Finalement, le domaine de l'attribut Saison est décomposé en quatre sous domaines : Dom(Saison) = $d_{31} \cup d_{32} \cup d_{33} \cup d_{34}$, avec d_{31} = {"Eté"}, d_{32} = {"Printemps"}, d_{33} = {"Automne"}, et d_{34} = {"Hiver"}.

La décomposition de chaque domaine d'attribut de fragmentation peut être représentée par un vecteur avec n_i cellules où n_i correspond au nombre de ses sous domaines. Les valeurs de ces cellules se situent entre 1 et n_i. En conséquence, chaque chromosome (schéma de fragmentation) peut être représenté par un tableau multidimensionnel. A partir de cette représentation, l'obtention des fragments horizontaux est simple. Elle consiste à générer toutes les clauses conjonctives.

Si deux cellules du même tableau contiennent la même valeur, alors elles seront fusionnées. Par exemple, dans la figure 1(b), nous pouvons déduire que la fragmentation de l'entrepôt de données n'est pas effectuée en utilisant l'attribut Sexe, parce que tous ses sous domaines ont la même valeur. En conséquence, l'entrepôt sera fragmenté en utilisant seulement les attributs Age et Saison. Pour l'attribut Saison, trois prédicats simples sont possibles : P_1 : Saison ='Eté', P_2 : Saison = 'Printemps' et P_3 : (Saison= 'Automne' $\cup$ Saison= 'Hiver'). Pour l'attribut Age, deux prédicats simples sont possibles, $P_4 = (Age \leq 18) \vee (Age \geq 60)$ et $P_5 : (18 < Age < 60)$. Par conséquent l'entrepôt de données peut être fragmenté en six fragments définis par les clauses suivantes : $Cl_1 : (P_1 \wedge P_4)$; $Cl_2 : (P_1 \wedge P_5)$; $Cl_3 : (P_2 \wedge P_4)$; $Cl_4 : (P_2 \wedge P_5)$; $Cl_5 : (P_3 \wedge P_4)$; et $Cl_6 : (P_3 \wedge P_5)$. Le codage que nous avons proposé satisfait les règles de correction (complétude, reconstruction et disjonction) (Bellatreche *et al.,* 2005). Il est utilisé pour représenter les fragments des tables de dimension et de la table de faits. Cette représentation en tableau multidimensionnel peut être utilisée pour générer tous les schémas de fragmentation possibles. Ce nombre est calculé par : $2^{(\sum_{i=1}^{K} n_i)}$ où K et n_i représentent le nombre d'attributs de fragmentation, et le nombre de sous domaines de l'attribut A_i, respectivement.

Si on considère les trois attributs de fragmentation, Age, Sexe et Saison (de l'exemple précédant), le nombre de tous les schémas possibles est $2^{(3+2+4)} = 2^9$. Ce nombre est pratiquement semblable au nombre de minterms possibles produits par l'algorithme de fragmentation horizontale primaire proposé par Ozsu et al. (Özsu *et al.,* 1999).

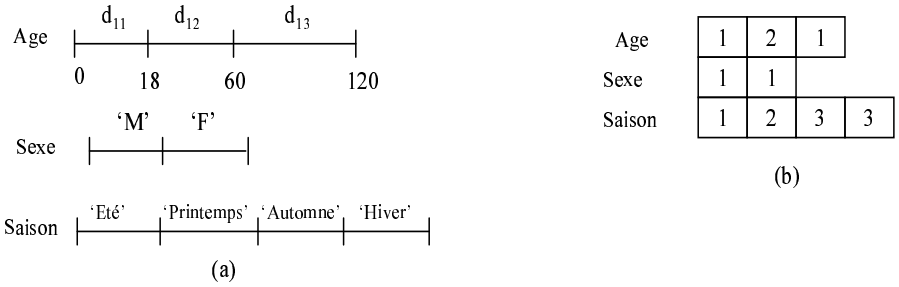

Figure 1. *(a) Sous domaines des attributs de fragmentation, (b) un exemple de schéma de fragmentation*

Pendant les processus de croisement et de mutation, les bons candidats survivent et les mauvais disparaissent. Dans notre travail, nous avons utilisé la méthode de la roulette (Bellatreche *et al.,* 2005). Chaque chromosome est associé à sa valeur de fitness calculée en utilisant un modèle de coût décrit dans la section 3.2. Les chromosomes avec une fitness élevée ont plus de chances d'être choisis. Nous avons utilisé un mécanisme de croisement multipoints pour éviter que les attributs ayant un nombre de sous domaines élevé (exemple de l'attribut Saison qui possède quatre sous domaines) aient une plus grande probabilité d'être croisés que des attributs avec un petit nombre de sous domaines, comme l'attribut Sexe (avec deux sous domaines).

3.2. *Optimisation avec contraintes : la fonction pénalité*

Chaque chromosome (schéma de fragmentation) FS_i généré par notre AG est évalué en utilisant une fonction de fitness représentée par un modèle de coût qui calcule la somme de pages accédées (entrées/sorties) nécessaires pour exécuter chaque requête dans l'entrepôt de données fragmenté. Notre modèle de coût suppose que toutes les tables de dimension sont stockées dans la mémoire centrale. Soit $D^{sel} = \{D_1^{sel}, ..., D_k^{sel}\}$ l'ensemble des tables de dimension ayant des attributs de sélection, où chaque prédicat de sélection p_j(défini sur une table de dimension D_i) possède un facteur de sélectivité noté par ($Sel_{D_i}^{p_j} \in [0, 1]$).

Pour chaque prédicat p_j, on définit son facteur de sélectivité dans la table des faits, noté par $Sel_F^{p_j}$ ($Sel_{D_i}^{p_j} \neq Sel_F^{p_j}$) (Bellatreche *et al.,* 2005). Dans notre étude, les facteurs de sélectivité sont choisis en utilisant une répartition uniforme (RU) et une répartition non uniforme (RNU).

Pour calculer notre fonction de fitness, chaque requête Q_k est exécutée sur chaque schéma FS_i $\{S_1, S_2, ..., S_{N_i}\}$. Afin d'identifier les sous schémas en étoile appropriés utilisés par cette requête, nous avons introduit une variable

booléenne dénotée par $V(Q_k, S_i)$ et définie par : $V(Q_k, S_i) = 1$ si le sous schéma S_i est utilisé par la requête Q_k, 0 sinon. Cependant, le coût d'exécution de la requête Q_k, en termes de nombre d'entrées sorties est donné par : $IOC(Q_k, FS_j) = (\sum_{j=1}^{N_i} V(Q_k, FS_j) \prod_{i=1}^{M_j} Sel_F^{p_i} \times \left\lceil \frac{||F|| \times L)}{PS} \right\rceil)$, avec, M_j, F, L et PS représentent le nombre de prédicats de sélection utilisés pour définir les fragments de sous schéma en étoile S_j, la cardinalité de la table des faits (nombre de tuples) F, la longueur, en bytes, de chaque tuple de la table des faits, et la taille d'une page disque (en bytes), respectivement. Le coût total pour exécuter l'ensemble des requêtes est donné par :

$$TIOC(Q, FS_i) = \sum_{k=1}^{m} IOC(Q_k, FS_i) \quad .$$

Notre problème de fragmentation d'un entrepôt de données relationnel consiste à partitionner un schéma en étoile, tel que le coût total d'exécution de requêtes soit minimal, ou en d'autres termes, le gain soit maximisé :
Maximiser $Eval(Q, FS_i) = (TIOC(Q, \phi) - TIOC(Q, FS_i))$ *tel que* $N_i \leq W$, avec $TIOC(Q, \phi)$ représente le coût d'exécution de l'ensemble de requêtes sur le schéma de l'entrepôt non fragmenté.

Notons que notre AG pourrait générer des schémas de fragmentation qui ne respectent pas la contrainte de maintenance (voir Section 2). Afin de pénaliser ces schémas, une fonction de pénalité est introduite et fait partie de la fonction de fitness. Dans cette étude, nous avons utilisé une fonction de pénalité linéaire (Yu *et al.*, 2004) définie comme suit : $Pen(Q, FS_i) = \alpha \times (N_i - W)$, avec α représente le facteur de pénalité (une constante > 0). Finalement, nous pouvons définir notre fonction de fitness finale suivant *le mode division* comme dans (Yu *et al.*, 2004) :

$$F(Q, FS_i) = \begin{cases} \frac{Eval(Q, FS_i)}{Pen(Q, FS_i)} & \text{si } Pen(Q, FS_i) > 1 \\ Eval(Q, FS_i) & \text{sinon.} \end{cases}$$

4. L'algorithme du recuit simulé

Le recuit simulé est appliqué sur la solution finale obtenue par notre AG (i.e., l'état initial du RS est le schéma de fragmentation généré par l'AG). Notons que cette solution est représentée par un tableau multidimensionnel (voir Figure 1b). Des modifications aléatoires sont appliquées sur ce tableau. Afin de faciliter l'exécution du RS, ce tableau est transformé en un tableau à une dimension, en concaténant toutes les lignes. Notons que les modifications aléatoires génèrent un nouveau problème, appelé 'validation d'une solution'. Cela est dû au fait que notre RS modifie les valeurs des cellules du tableau en les incrémentant ou les décrémentant. Ceci peut causer un débordement des valeurs en dehors du domaine des cellules. Pour résoudre ce problème nous avons développé une technique qui vérifie la validité de chaque solution générée par le RS.

La valeur de la fonction fitness de chaque solution est calculée en utilisant le modèle de coût développé dans la section 3.2. Les différentes étapes de notre algorithme RS sont décrites dans Algorithme 1.

Algorithm 1 Algorithme Recuit Simulé

Entrées : EtatInitial (représentant le schéma de fragmentation obtenu par l'AG) ;
TempératureInitiale ;
Sortie : EtatMin ;
début
EtatMin := EtatInitial ; coût := **F(Q, EtatInitial)** ; CoûtMin := Coût ;
temp := TempératureInitiale ;
répéter
répéter
NouveauEtat := Transformation(EtatMin) ; *TestValidité(NouveauEtat)* ;
NouveauCoût := F(Q, NouveauEtat) ;
si (NouveauCoût $\leq$ Coût) **alors**
EtatCourant := NouveauEtat ; Coût := NouveauCoût ;
sinon avec une probabilité $e^{\frac{NouveauCot-Cot}{temp}}$
EtatCourant := NouveauEtat ; Coût := NouveauCoût ;
fin ;
si (Coût $<$ CoûtMin) **alors** EtatMin := EtatCourant ; CoûtMin := Coût ;
fin
jusqu'à l'obtention d'un équilibre ;
réduire la température ;
jusqu'à gel du système ;
Retourner EtatMin
fin

5. Etudes expérimentales

Pour notre étude, nous avons utilisé le Benchmark APB1(Olap Council. 98). Le schéma en étoile de ce benchmark possède une table de faits Actvars(Customer_level, Product_level, Channel_level, Time_level, UnitsSold, DollarSales) (24 786 000 tuples), et quatre tables de dimension : Prodlevel (Code_level, Class_level, Group_level, Family_level, Line_level, Division_level)(9 000 tuples), Custlevel (Store_level, Retailer_level)(900 tuples), Timelevel (Tid, Year_level, Quarter_level, Month_level) (24 tuples), Chanlevel (Base_level, All_level) (9 tuples).

Cet entrepôt de données est alimenté en utilisant le module de génération fournis avec ce benchmark. Chaque table de dimension peut être jointe avec la table de faits à travers son premier attribut. Notre programme de simulation est implémenté en utilisant Visual C exécuté sous PC Pentium IV, d'une fréquence de 1,5 Ghz et une mémoire de 256 Mo.

5.1. *Identification des paramètres des algorithmes*

Pour mener notre évaluation, nous avons considéré un ensemble de 15 requêtes, chaque requête possède des prédicats de sélection, où chacun est associé à un facteur de sélectivité. La taille de la page système est de 65536 Octets. Nous considérons

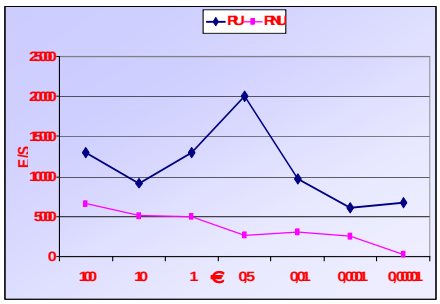

Figure 2. *Effet du facteur de pénalité sur la qualité des schémas de fragmentation*

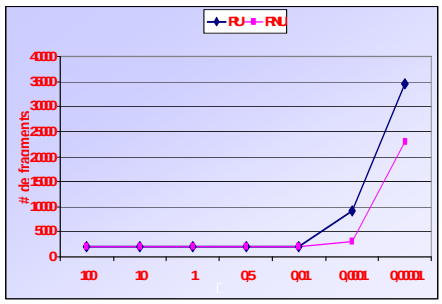

Figure 3. *Etude de l'impact du facteur de pénalité sur le nombre de fragments*

9 attributs de fragmentation. Le nombre de sous domaines générés par ces attributs est 40. Un algorithme exhaustif doit générer 2^{40} schémas de fragmentations différents pour trouver la solution optimale. En raison de ce nombre important, nous n'avons pas considéré la recherche exhaustive. Nos expérimentations se sont déroulées de la façon suivante : (i) identification des bons paramètres de notre AG qui sont : le nombre de générations, le nombre de chromosomes par génération, le taux de croisement, le taux de mutation, (ii) exécution de notre AG en utilisant ces paramètres, et finalement, (3) exécution de RS sur la solution obtenue par notre AG.

Les bons paramètres obtenus dans la première expérimentation sont : 500 générations, 40 chromosomes par génération, un taux de croisement de 75%, un taux de mutation de 30% au début, après plusieurs générations, un taux de mutation de 6% est utilisé pour éviter une recherche superflue.

Nous avons réalisé des expérimentations pour avoir la meilleure valeur du facteur de pénalité α. Pour se faire, nous avons choisi plusieurs valeurs de α et pour chaque valeur, nous avons exécuté notre AG. Les résultats obtenus montrent que le facteur de pénalité α a un effet considérable sur la solution finale.

La figure 2 montre que lorsque α est petit, la performance des requêtes est bonne. La logique derrière ce résultat est que quand α est petit, la contrainte de maintenance est moins prise en considération, donc notre AG explore plus de solutions et n'ignore pas les solutions pénalisées. La figure 3 confirme ces résultats et montre l'explosion du nombre de fragments quand α est en dessous de 0,01 pour une répartition non uniforme. Pour le reste des expérimentations nous avons choisi $\alpha = 0,01$.

5.2. *Expérimentations de l'algorithme génétique*

Afin d'évaluer l'impact des attributs de fragmentation utilisés, nous avons fait une expérience en changeant le nombre de ces attributs. Nous commençons de 1 à 9 at-

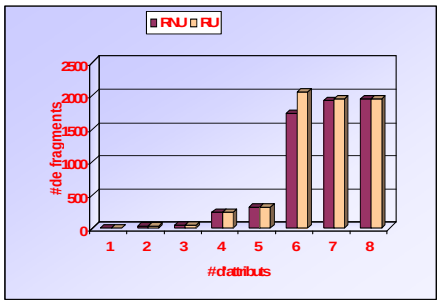

Figure 4. *Impact du nombre d'attributs de fragmentation sur le nombre final de fragments*

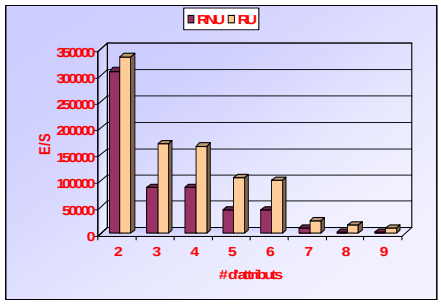

Figure 5. *Impact du nombre d'attributs de fragmentation sur le nombre d'E/S total*

tributs, et pour chaque valeur, nous calculons le nombre de fragments générés et le nombre d'E/S pour exécuter les 15 requêtes.

Les résultats obtenus montrent que le nombre de fragments et d'E/S sont proportionnels au nombre d'attributs de fragmentation utilisés (figures 4, 5). La principale directive issue de cette étude que le DWA pourrait prendre en considération s'il veut accélérer ses requêtes, est l'utilisation d'un grand nombre d'attributs de fragmentation couvrant toutes les tables de dimension. Cela est dû au fait que les requêtes OLAP sont souvent exécutées sur la totalité du schéma en étoile. La figure 6 montre l'effet des tables de dimension participant dans le processus de fragmentation. La performance des requêtes OLAP est proportionnelle au nombre de ces tables.

Dans la figure 7, nous avons constaté que le rapport entre le nombre de fragments retourné par notre AG et le nombre de fragments possibles, varie selon le nombre d'attributs de fragmentation utilisés. Quand six attributs sont utilisés, ce rapport est grand. Au-delà de six attributs, ce rapport devient très petit (inférieur à 1,5%). Cela est dû à l'augmentation du nombre de fragments possibles. Notons que le nombre de fragments augmente quand le seuil augmente, mais reste très proche de ce dernier (figure 8).

Dans la figure 9, nous étudions l'effet des facteurs de sélectivité des prédicats de sélection utilisés dans le processus de fragmentation sur le nombre de fragments et la performance des requêtes. Nous en avons conclu que l'augmentation des valeurs de ces facteurs génère l'augmentation du nombre d'E/S. Cela est dû au fait qu'une haute sélectivité implique un grand nombre de tuples satisfaisant les prédicats. Notons que les facteurs de sélectivité n'ont pas d'effets importants sur le nombre de fragments finaux.

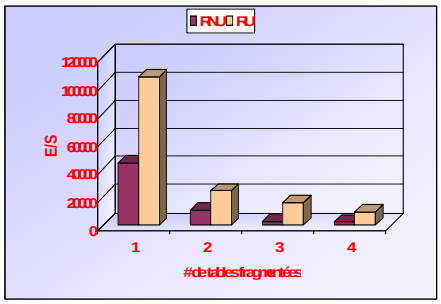

Figure 6. *Nombre de tables de dimension participant dans le processus de fragmentation et leurs effets sur la performance*

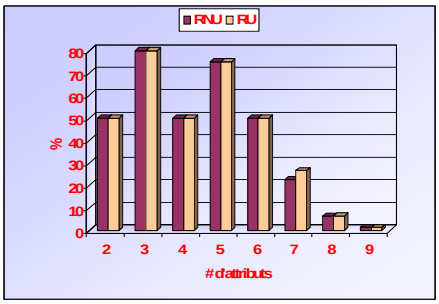

Figure 7. *Rapport entre le nombre de fragments générés et les fragments possibles*

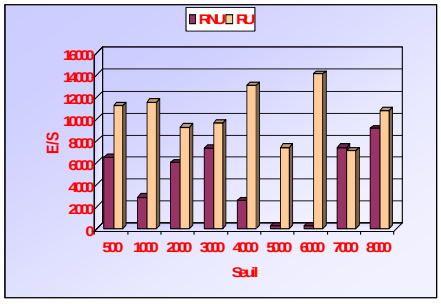

Figure 8. *Evolution du nombre D'E/S par rapport au seuil*

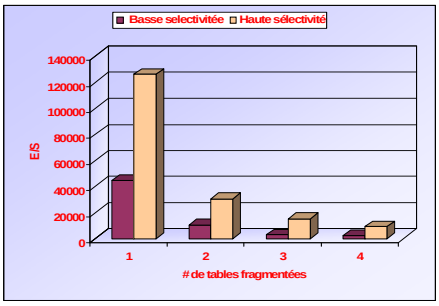

Figure 9. *Effet des facteurs de sélectivité sur la performance des requêtes*

5.3. *Expérimentations du recuit simulé*

Les paramètres utilisés dans notre RS sont : température initiale 400 décrémentée de 2 chaque 100 itérations, seuil 2000, α est 0,01 et le nombre d'itérations sans amélioration est fixé à 21000. Nous avons étudié l'effet de AG et RS sur les différentes requêtes. La réduction du coût est significative lorsque nous utilisons RS.

Nous avons remarqué que parmi les 15 requêtes, certaines n'ont pas bénéficiées de l'application du RS, car leurs prédicats de sélection ne coïncident pas avec les prédicats des fragments générés par le RS (voir figure 10). Finalement la figure 11, montre l'impact du RS sur la réduction du coût global d'exécution des requêtes. Il est réduit de 44% après l'application du RS. Nous pouvons donc en conclure que la combinaison de l'AG et le RS donne de bons résultats (le RS est complémentaire à l'AG).

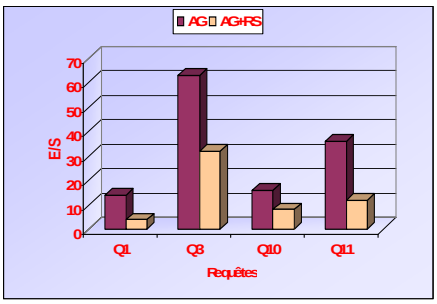

Figure 10. *Requêtes bénéficiaires*

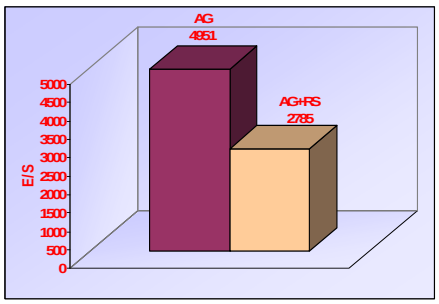

Figure 11. *Réduction d'E/S après l'application du RS*

6. Conclusion

La conception physique des entrepôts de données inclut les vues matérialisées, l'indexation et la fragmentation des données. Dans ce travail, nous nous sommes concentrés sur la fragmentation horizontale des entrepôts de données relationnels. Nous avons formalisé ce problème comme un problème d'optimisation et nous avons ainsi étudié sa complexité. Le nombre de fragments de faits générés par notre procédure de fragmentation peut être très grand et il devient très difficile à DWA de gérer tous les fragments. Pour résoudre ce problème de fragmentation des données, nous avons proposé une approche hybride combinant les AGs et le RS. Les AGs sont bons pour l'optimisation des problèmes avec un très grand espace de recherche. Le RS est appliqué sur la solution obtenue par l'AG afin d'éviter le problème de convergence prématurée inhérente aux AGs en permettant des déplacements aux solutions de mauvaises fitness. Cette fitness est caractérisée par un modèle de coût qui évalue le coût d'exécution d'un ensemble de requêtes fréquentes exécutées sur un entrepôt de données relationnel modélisé par un schéma en étoile. La fonction fitness est combinée avec une valeur de pénalité. La solution que nous proposons a été implémentée en utilisant le benchmark APB1. Pour la valider nous avons suivi trois étapes : (1) des expérimentations pour identifier les bons paramètres de notre algorithme génétique, (2) des expérimentations basées sur ces paramètres, et (3) finalement, des expériences en combinant les AGs et le RS. Les résultats expérimentaux sont encourageants et montrent la faisabilité de notre approche.

Pour les travaux futurs, il serait intéressant d'adapter les algorithmes que nous avons proposés dans ce travail afin de sélectionner des index de jointure compte tenu de la similitude entre les deux structures d'optimisation.

7. Bibliographie

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

Bellatreche L., K. B., « An Evolutionary Approach to Schema Partitioning Selection in a Data Warehouse Environment », *Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2005)*, vol. , p. 115-125, August, 2005.

Bellatreche L., Schneider M., Lorinquer H., Mohania M., « 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)*, vol. , p. 15-25, September, 2004.

Bennett K. P., Ferris M. C., Ioannidis Y. E., « A Genetic Algorithm for Database Query Optimization », *in Proceedings of the 4th International Conference on Genetic Algorithms*, vol. , p. 400-407, July, 1991.

Chaudhuri S., Narasayya V., « AutoAdmin 'What-if' Index Analysis Utility », *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. , p. 367-378, June, 1998.

Holland J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, 1975.

Kalnis P., Papadias D., « Proxy-Server Architecture for OLAP », *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2001.

Kirkpatrick S., Gelatt C. D., Vecchi M. P., « Optimization by Simulated Annealing », *Science*, vol. 220, n° 4598, p. 671-680, May, 1983.

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

Papadomanolakis S., Ailamaki A., « AutoPart : Automating Schema Design for Large Scientific Databases Using Data Partitioning », *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004)*, vol. , p. 383-392, June, 2004.

Rao J., Zhang C., Lohman G., Megiddo N., « Automating Physical Database design in a parallel database », *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. , p. 558-569, June, 2002.

Sanjay A., Narasayya V. R., Yang B., « Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design », *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. , p. 359-370, June, 2004.

Stöhr T., Märtens H., Rahm E., « Multi-Dimensional Database Allocation for Parallel Data Warehouses », *Proceedings of the International Conference on Very Large Databases*, vol. , p. 273-284, 2000.

Yang J., Karlapalem K., Li Q., « Algorithms for materialized view design in data warehousing environment », *Proceedings of the International Conference on Very Large Databases*, vol. , p. 136-145, August, 1997.

Yu J. X., Choi C.-H., Gou G., « Materialized View Selection as Constrained Evolution Optimization », *IEEE Transactions On Systems, Man, and Cybernetics, Part 3*, vol. 33, n° 4, p. 458-467, November, 2004.